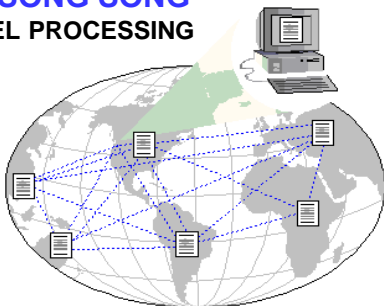


## XỬ LÝ SONG SONG PARALLEL PROCESSING



### THÔNG TIN VỀ HỌC PHẦN

#### 2. Mục tiêu của học phần:

2.1. **Kiến thức:** cung cấp cho người học các kiến thức về máy tính song song, cách xây dựng các thuật toán song song.

2.2. **Kỹ năng:** sử dụng công cụ lập trình song song như MPI, JAVA, VPM ... người học phải cài đặt được một số thuật toán song song cơ bản.

2.3. **Thái độ học tập:** người học phải tham dự đầy đủ các giờ lý thuyết và thảo luận.

### THÔNG TIN VỀ HỌC PHẦN

#### 3. Phân bố số tiết:

- Lý thuyết: 20
- Tiểu luận, đọc thêm: 8
- Thảo luận: 2

### THÔNG TIN VỀ HỌC PHẦN

#### 1. Thông tin chung:

- 1.1. Tên học phần: **Xử lý song song** (Parallel Processing)
- 1.2. Mã học phần: **KH.KM.515**
- 1.3. Số tín chỉ: 2 TC
- 1.4. Loại học phần: **Bắt buộc**
- 1.5. Các học phần tiên quyết: **Cơ sở dữ liệu phân tán**

### THÔNG TIN VỀ HỌC PHẦN

#### 3. Chính sách đối với học phần

•**Tham gia học tập trên lớp:** đi học đầy đủ, tích cực thảo luận nội dung bài giảng, tham gia chữa bài tập và chuẩn bị bài vở tốt - 20%.

•**Khả năng tự học, tự nghiên cứu:** hoàn thành bài tiểu luận (assignment) theo từng cá nhân. Bài kiểm tra đánh giá giữa kỳ - 20%.

•**Kết quả thi cuối kỳ** - 60%.

### NỘI DUNG CHƯƠNG TRÌNH

#### PHẦN 1: TÍNH TOÁN SONG SONG

- |          |  |
|----------|--|
| Chương 1 | KIẾN TRÚC VÀ CÁC LOẠI MÁY TINH SONG SONG |
| Chương 2 | CÁC THÀNH PHẦN CỦA MÁY TINH SONG SONG    |
| Chương 3 | GIỚI THIỆU VỀ LẬP TRÌNH SONG SONG        |
| Chương 4 | CÁC MÔ HÌNH LẬP TRÌNH SONG SONG          |
| Chương 5 | THUẬT TOÁN SONG SONG                     |

#### PHẦN 2: XỬ LÝ SONG SONG CÁC CƠ SỞ DỮ LIỆU

(Đọc thêm)

- |          |  |
|----------|--|
| Chương 6 | TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU SONG SONG       |
| Chương 7 | TỐI ƯU HÓA TRUY VẤN SONG SONG              |
| Chương 8 | LẬP LỊCH TỐI ƯU CHO CÂU TRUY VẤN SONG SONG |

## TÀI LIỆU THAM KHẢO

- [0] Đoàn văn Ban, Nguyễn Mậu Hân, *Xử lý song song và phân tán*, NXB KH&KT, 2006.
- [1] Ananth Grama, Anshul Gupta, George Karpis, Vipin Kumar, *Introduction to Parallel Computing*, Pearson, 2003
- [2] Barry Wilkinson, Michael Allen, *Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall New Jersey, 1999
- [3] M. Sasikumar, Dinesh Shikhar, P. Ravi Prakash, *Introduction to Parallel Processing*, Prentice - Hall, 2000
- [4] Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation*, Springer 1999.
- [5] Michael J. Quinn, *Parallel Computing Theory and Practice*, McGraw-Hill, 1994
- [6] Shaharuddin Salleh, Albert Y. Zomaya, *Scheduling in Parallel Computing Systems*, Kluwer Academic Publisher, 1999.

## TÀI LIỆU THAM KHẢO

- [7] Clement T.Yu, Weiyei Meng, *Principles of Database Query Processing for Advanced Applications*, Morgan Kaufman Inc., 1998. 185-225.
- [8] Hasan Waqar, *Optimization of SQL Query for Parallel Machines*, Springer, 1995.
- [9] Hong W., *Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays*, University of California, 1992.
- [10] Hua, K.A., *Parallel Database Technology*, University of Central Florida Orlando FL 32846-2362, 1997.
- [11] Zomaya A. Y. and Shaharuddin Salleh, *Scheduling in parallel computing systems*, Kluwer Academic Publishers, 1999.

## ĐỊA CHỈ LIÊN HỆ

TS. NGUYỄN MẬU HÂN  
KHOA CÔNG NGHỆ THÔNG TIN  
TRƯỜNG ĐẠI HỌC KHOA HỌC - ĐẠI HỌC HUẾ  
77, NGUYỄN HUỆ - HUẾ

DIỆN THOẠI:

CQ: 054 382 6767

ĐD: 01255213579

EMAIL: [nmhan2005@yahoo.com](mailto:nmhan2005@yahoo.com)

[back](#)



## CHƯƠNG 1. KIẾN TRÚC CÁC LOẠI MÁY TÍNH SONG SONG

### NỘI DUNG

#### 1.1 Giới thiệu chung

#### 1.2 Kiến trúc máy tính kiểu Von Neumann

#### 1.3 Phân loại máy tính song song

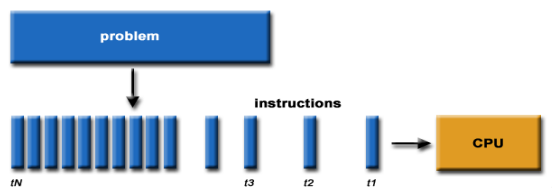
#### 1.4 Kiến trúc máy tính song song

### 1.1 Giới thiệu chung

*Xử lý song song (XLSS) là gì?*

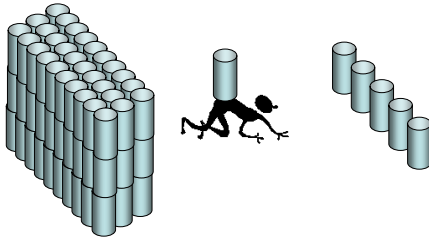
*Trong xử lý tuần tự:*

- Bài toán được tách thành một chuỗi các câu lệnh rời rạc
- Các câu lệnh được thực hiện một cách tuần tự
- Tại mỗi thời điểm chỉ thực hiện được một câu lệnh



## 1.1 Giới thiệu chung (tt)

1 CPU  
Đơn giản  
Chậm quá !!!

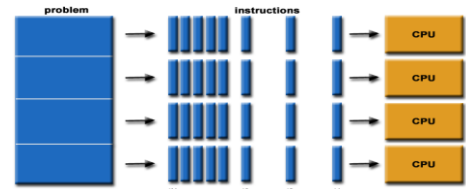


13

## 1.1 Giới thiệu chung (tt)

*Trong xử lý song song*

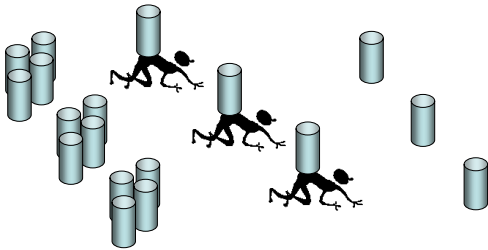
- Bài toán được tách thành nhiều phần và có thể thực hiện đồng thời.
- Mỗi phần được tách thành các lệnh rời rạc
- Mỗi lệnh được thực hiện từ những CPU khác nhau



14

## 1.1 Giới thiệu chung (tt)

Nhiều CPU  
Phức tạp hơn  
Nhanh hơn !!!



15

## 1.1 Giới thiệu chung (tt)

*Vậy xử lý song song là gì?*



XLSS là một quá trình xử lý gồm nhiều tiến trình được kích hoạt đồng thời và cùng tham gia giải quyết một vấn đề trên hệ thống có nhiều bộ xử lý.

16

## 1.1 Giới thiệu chung (tt)

**Tại 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

17

## 1.1 Giới thiệu chung (tt)

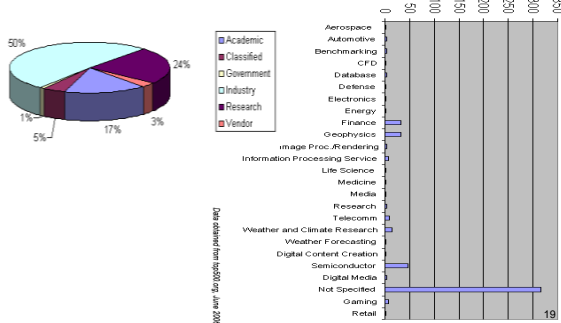
*Sự khác nhau cơ bản giữa XLSS và XLTT :*

Xử lý tuần tự	Xử lý song song
Mỗi thời điểm chỉ thực hiện được <b>một</b> phép toán	Mỗi thời điểm có thể thực hiện được <b>nhiều</b> phép toán
Thời gian thực hiện phép toán <b>chậm</b>	Thời gian thực hiện phép toán <b>nhanh</b>

18

## 1.1 Giới thiệu chung (tt)

### Đối tượng nào sử dụng máy tính song song?



## 1.1 Giới thiệu chung (tt)

### Những thành phần liên quan đến vấn đề XLSS:

- ☐ Kiến trúc máy tính song song
- ☐ Phần mềm hệ thống (hệ điều hành),
- ☐ Thuật toán song song
- ☐ Ngôn ngữ lập trình song song, v.v.

### Định nghĩa máy tính song song (MTSS):

MTSS là một tập các BXL (thường là cùng một loại) kết nối với nhau theo một kiểu nào đó để có thể **hợp tác với nhau cùng hoạt động và trao đổi dữ liệu** với nhau.

21

## 1.1 Giới thiệu chung (tt)

### Tính khả thi của việc XLSS?

- Tốc độ xử lý của các BXL theo kiểu Von Neumann bị giới hạn, không thể cải tiến thêm được.
- Giá thành của phần cứng (CPU) giảm, tạo điều kiện để xây dựng những hệ thống có nhiều BXL với giá cả hợp lý.
- Sự phát triển công nghệ mạch tích hợp cao VLSI (very large scale integration) cho phép tạo ra những hệ phức hợp có hàng triệu transistor trên một chip.

20

## 1.1 Giới thiệu chung (tt)

### Tiêu chí để đánh giá một thuật toán song song

#### Đối với thuật toán tuần tự

- thời gian thực hiện thuật toán.
- không gian bộ nhớ.
- khả năng lập trình.

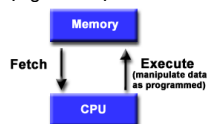
#### Đối với thuật toán song song

- 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.

22

## 1.2 Kiến trúc máy tính kiểu Von Neumann

- John von Neumann (1903–1957): nhà toán học người Hungary
- Sử dụng khái niệm lưu trữ chương trình (stored-program concept)



### Von Neumann computer có các đặc điểm sau:

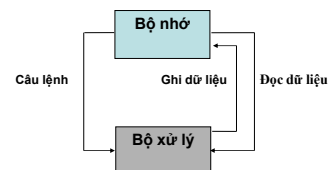
- Bộ nhớ được dùng để lưu trữ chương trình và dữ liệu
- Chương trình được mã hoá (code) để máy tính có thể hiểu được
- Dữ liệu là những thông tin đơn giản được sử dụng bởi chương trình
- CPU nạp (fetch) những lệnh và dữ liệu từ bộ nhớ, giải mã (decode) và thực hiện tuần tự chúng.

23

## 1.2 Kiến trúc máy tính kiểu Von Neumann

Máy tính kiểu V. Neumann được xây dựng từ các khối cơ sở:

- **Bộ nhớ:** để lưu trữ dữ liệu
- **Các đơn vị logic và số học ALU:** thực hiện các phép toán
- **Các phần tử xử lý:** điều khiển CU và truyền dữ liệu I/O
- **Đường truyền dữ liệu: BUS**



24

### 1.3 Phân loại máy tính song song

#### □ Tiêu chí để phân loại máy tính song song?

a) **Dựa trên lệnh, dòng dữ liệu và cấu trúc bộ nhớ (Flynn)**

b) **Dựa trên kiến trúc: (xem 1.4)**

- Pipelined Computers
- Dataflow Architectures
- Data Parallel Systems
- Multiprocessors
- Multicomputers

25

### 1.3 Phân loại máy tính song song

#### □ Michael Flynn (1966)

- **SISD**: **S**ingle **I**nstruction Stream, **S**ingle **D**ata Stream  
Đơn luồng lệnh, đơn luồng dữ liệu
- **SIMD**: **S**ingle **I**nstruction Stream, **M**ultiple **D**ata Stream  
Đơn luồng lệnh, đa luồng dữ liệu
- **MISD**: **M**ultiple **I**nstruction Stream, **S**ingle **D**ata Stream  
Đa luồng lệnh, đơn luồng dữ liệu
- **MIMD**: **M**ultiple **I**nstruction Stream, **M**ultiple **D**ata Stream  
Đa luồng lệnh, đa luồng dữ liệu

26

### 1.3 Phân loại máy tính song song

**Mô hình SISD - Đơn luồng lệnh, đơn luồng dữ liệu**

#### Đặc điểm

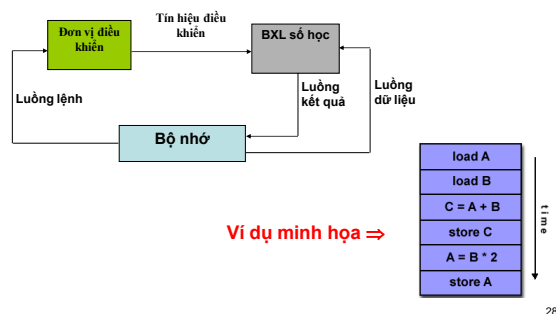
- Chỉ có một CPU
- Ở mỗi thời điểm chỉ thực hiện một lệnh và chỉ đọc/ghi một mục dữ liệu
- Có một thanh ghi, gọi là bộ đếm chương trình (*program counter*), được sử dụng để nạp địa chỉ của lệnh tiếp theo khi xử lý tuần tự
- Các câu lệnh được thực hiện theo một thứ tự xác định

*Đây chính là mô hình máy tính truyền thống kiểu Von Neumann*

27

### 1.3 Phân loại máy tính song song

**Mô hình SISD - Đơn luồng lệnh, đơn luồng dữ liệu (tt)**



28

### 1.3 Phân loại máy tính song song

**Mô hình SIMD - Đơn luồng lệnh, đa luồng dữ liệu**

- Có một đơn vị điều khiển (CU) để điều khiển nhiều đơn vị xử lý (PE)
- CU phát sinh tín hiệu điều khiển đến các đơn vị xử lý
- **Đơn luồng lệnh**: các đơn vị xử lý thực hiện cùng một lệnh trên các mục dữ liệu khác nhau
- **Đa luồng dữ liệu**: mỗi đơn vị xử lý có luồng dữ liệu riêng
- Đây là kiểu tính toán lặp lại các đơn vị số học trong CPU, cho phép những đơn vị khác nhau thực hiện trên những toán hạng khác nhau, nhưng thực hiện cùng một lệnh.
- Máy tính SIMD có thể hỗ trợ xử lý kiểu vector, trong đó có thể gán các phần tử của vector cho các phần tử xử lý để tính toán đồng thời.

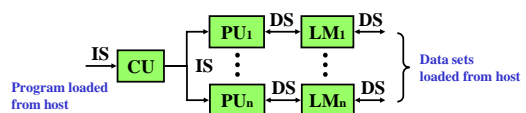
29

### 1.3 Phân loại máy tính song song

**Mô hình SIMD - Đơn luồng lệnh, đa luồng dữ liệu (tt)**

IS: Instruction Stream  
LM: Local Memory

PU: Processing Unit  
DS: Data Stream

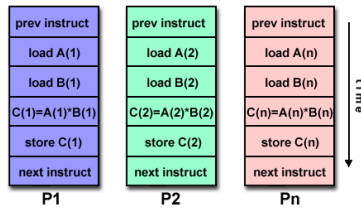


Mô hình của kiến trúc SIMD với bộ nhớ phân tán

30

### 1.3 Phân loại máy tính song song

#### Mô hình SIMD - Đa luồng lệnh, đa luồng dữ liệu (tt)



Các máy tính trên thị trường được sản xuất theo mô hình SIMD: *ILLIAC IV*, *DAP* và *Connection Machine CM-2*

31

### 1.3 Phân loại máy tính song song

#### Mô hình MISD - Đa luồng lệnh, đơn luồng dữ liệu

##### Đặc điểm:

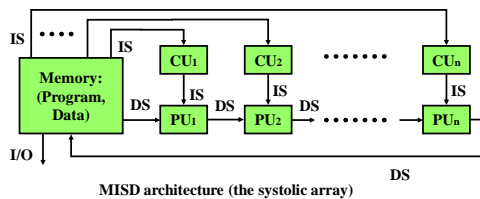
- **Đa luồng lệnh:** có thể thực hiện nhiều lệnh trên cùng một mục dữ liệu
- **Đơn luồng dữ liệu:** các PU xử lý trên cùng một luồng dữ liệu
- **Kiến trúc kiểu này có thể chia thành hai nhóm:**
  - ❑ Các máy tính yêu cầu mỗi đơn vị xử lý (PU) nhận những lệnh khác nhau để thực hiện trên cùng một mục dữ liệu.
  - ❑ Các máy tính có các luồng dữ liệu được chuyển tuần tự theo dây các CPU liên tiếp-gọi là kiến trúc hình ống-xử lý theo vector thông qua một dây các bước, trong đó mỗi bước thực hiện một chức năng và sau đó chuyển kết quả cho PU thực hiện bước tiếp theo.

32

### 1.3 Phân loại máy tính song song

#### Mô hình MISD – Đa luồng lệnh, Đơn luồng dữ liệu (tt)

IS: Instruction Stream    PU: Processing Unit    CU: Control Unit  
LM: Local Memory    DS: Data Stream

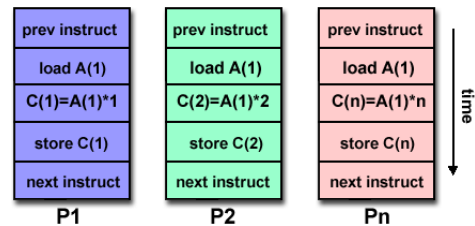


33

### 1.3 Phân loại máy tính song song

#### Mô hình MISD – Đa luồng lệnh, Đơn luồng dữ liệu (tt)

##### Ví dụ minh họa



34

### 1.3 Phân loại máy tính song song

#### Mô hình MIMD - Đa luồng lệnh, đa luồng dữ liệu

- 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 sự trao đổi giữa các BXL trong hệ thống.

##### Nhận xét:

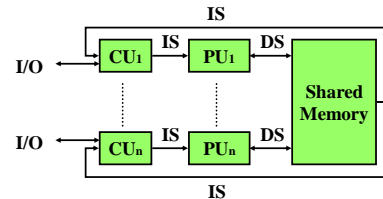
- Đây là kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất
- Các máy tính được sản xuất theo kiến trúc này:

*BBN Butterfly*, *Alliant FX*, *iSPC của Intel*

35

### 1.3 Phân loại máy tính song song

#### Mô hình MIMD – Đa luồng lệnh, Đa luồng dữ liệu (tt)

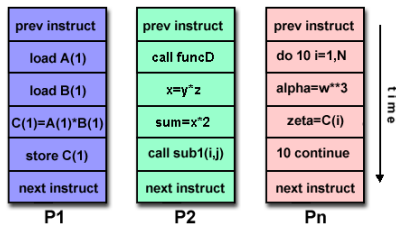


MIMD architecture with shared memory

36

### 1.3 Phân loại máy tính song song

#### Mô hình MIMD – Đa luồng lệnh, Đa luồng dữ liệu (tt)



#### Nhận xét:

•MIMD là kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất

•Các máy tính được sản xuất theo kiến trúc này:

BBN Butterfly, Alliant FX, iSPC của Intel

37

### 1.4 Kiến trúc máy tính song song

#### Một vài nhận xét:

- Theo Flynn: có hai họ kiến trúc quan trọng cho các máy tính song song: **SIMD** và **MIMD**. Những kiến trúc khác có thể xếp theo hai mẫu đó.
- Mục tiêu** của xử lý song song là *khai thác đến mức tối đa các khả năng sử dụng của các thiết bị phần cứng* nhằm giải quyết nhanh những bài toán đặt ra trong thực tế.
- Kiến trúc phần cứng là **trọng yếu** đối với người lập trình
- Trong kiến trúc tuần tự có thể tận dụng tốc độ cực nhanh của BXL để thực hiện xử lý song song theo nguyên lý **chia sẻ thời gian** và **chia sẻ tài nguyên**.
- Những chương trình song song trên máy đơn BXL có thể thực hiện được nếu có HĐH cho phép nhiều tiến trình cùng thực hiện, nghĩa là có thể xem hệ thống như là đa bộ xử lý.

38

### 1.4 Kiến trúc máy tính song song

#### Song song hóa trong máy tính tuần tự:

##### a. Đa đơn vị chức năng:

- Các máy tính truyền thống chỉ có một đơn vị số học và logic (ALU) trong BXL. Ở mỗi thời điểm nó chỉ có thể thực hiện một chức năng.
- Máy tính song song có nhiều đơn vị xử lý (PE). Những đơn vị này có thể cùng nhau thực hiện song song.  
Ví dụ: máy CDC 6600 (1964) có 10 PE được tổ chức trong một BXL. Những đơn vị chức năng này độc lập với nhau và có thể thực hiện đồng thời.
- Xây dựng **bộ lập lịch tối ưu** để phân chia các câu lệnh thực hiện sao cho tận dụng được tối đa các đơn vị xử lý cũng như các tài nguyên của máy tính.

39

### 1.4 Kiến trúc máy tính song song

#### Song song hóa trong máy tính tuần tự (tt):

##### b. Xử lý theo nguyên lý hình ống trong CPU

- Câu lệnh được chia thành các giai đoạn (stage-phase)
- Tại một thời điểm có thể có nhiều lệnh được tải vào và được thực hiện trong những bước khác nhau
- Các giai đoạn thực hiện khác nhau của mỗi câu lệnh có thể thực hiện gối đầu nhau.
- Đầu ra của giai đoạn này có thể là đầu vào của giai đoạn tiếp theo
- Thực hiện theo nguyên lý hình ống sẽ hiệu quả hơn vì không cần vùng đệm dữ liệu.

40

### 1.4 Kiến trúc máy tính song song

#### Song song hóa trong máy tính tuần tự (tt):

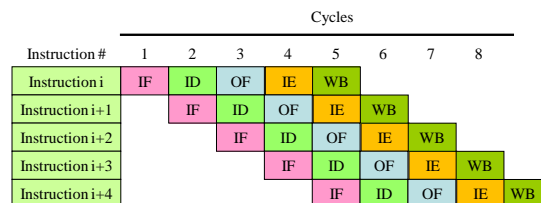
##### b. Xử lý theo nguyên lý hình ống trong CPU

- Ví dụ:
  - Pha 1: nạp câu lệnh về từ bộ nhớ (Instruction Fetch)
  - Pha 2: giải mã (Instruction decode)
  - Pha 3: xác định các toán hạng (Operand Fetch)
  - Pha 4: thực hiện câu lệnh (Instruction Execute)
  - Pha 5: lưu trữ kết quả (Write-Back)
- ... quá trình này có thể phân cho mỗi PE thực hiện một công việc. Theo cách đó, tại một thời điểm BXL có thể thực hiện được nhiều câu lệnh gối đầu nhau. Trước khi một câu lệnh kết thúc thực hiện thì câu lệnh tiếp theo đã có thể thực hiện pha giải mã, câu lệnh khác lại có thể được nạp về, v.v.

41

### 1.4 Kiến trúc máy tính song song

IF: Instruction Fetch  
ID: Instruction decode  
OF: Operand Fetch  
IE: Instruction Execute  
WB: Write-Back

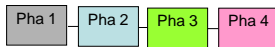


42

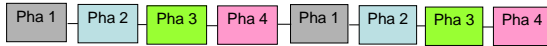
## 1.4 Kiến trúc máy tính song song

Ví dụ: Thực hiện tuần tự và hình ống của 2 tiến trình gồm 4 giai đoạn

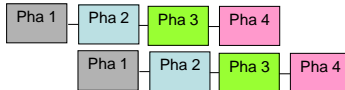
▪ Giả sử một tiến trình được chia thành 4 giai đoạn:



▪ Thực hiện tuần tự 2 tiến trình phải qua 8 giai đoạn:



▪ Thực hiện theo hình ống chỉ cần trải qua 5 giai đoạn:



Tổng thời gian tính toán tuần tự là:  $2 * (S_1 + S_2 + S_3 + S_4)$

Tổng thời gian tính toán hình ống là:  $S_1 + S_2 + S_3 + S_4 + S_4$

43

## 1.4 Kiến trúc máy tính song song

**Song song hóa trong máy tính tuần tự (tt):**

**c. Sự gởi đầu CPU và các thao tác vào/ra (I/O)**

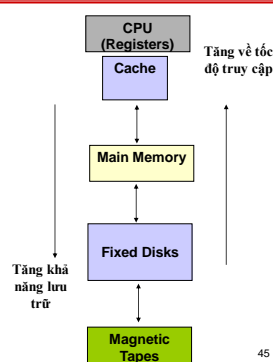
- Các phép vào/ra có thể thực hiện đồng thời đối với nhiều nhiệm vụ tính toán khác nhau bằng cách sử dụng những bộ điều khiển vào/ra, các kênh hay những BXL vào/ra khác nhau.
- Nhiều máy tính hiện nay có nhiều bộ điều khiển thiết bị vào/ra, cho phép đa xử lý vào/ra làm tăng được tốc độ trao đổi dữ liệu giữa các thiết bị ngoài với CPU.

44

## 1.4 Kiến trúc máy tính song song

**d. Các hệ thống bộ nhớ phân cấp**

- Do tốc độ thực hiện các phép toán trong BXL nhanh hơn rất nhiều việc đọc dữ liệu vào bộ nhớ trong
- Các thanh ghi được sử dụng trực tiếp cho ALU nên bộ nhớ cache được xem như vùng đệm giữa BXL và bộ nhớ chính
- Khi dữ liệu được chuyển từ bộ nhớ cache vào bộ nhớ chính thì đồng thời có thể chuyển dữ liệu từ cache vào cho CPU xử lý



45

## 1.4 Kiến trúc máy tính song song

**Song song hóa trong máy tính tuần tự (tt):**

**e. Đa chương trình và chia sẻ thời gian**

- Thực hiện song song dựa vào hệ điều hành đa nhiệm, phần mềm đa luồng, đa tiến trình.
- Hệ điều hành đa nhiệm thường giải quyết các trường hợp:
  - Trong cùng một khoảng thời gian, có nhiều tiến trình cùng truy cập vào dữ liệu từ thiết bị vào/ra chung
  - Một tiến trình tính toán với cường độ cao có thể tạm thời chiếm dụng CPU để làm việc, trong khi một tiến trình khác trước đó không đòi hỏi phải kết thúc công việc sớm phải ngưng lại.
  - **Bộ lập lịch** chia sẻ thời gian làm nhiệm vụ phân chia CPU cho mỗi tiến trình một khoảng thời gian cố định
  - Tạo các **BXL ảo**: mỗi tiến trình được cung cấp một môi trường được xem như một BXL để thực hiện riêng cho tiến trình đó.

46

## 1.4 Kiến trúc máy tính song song

**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.

47

## 1.4 Kiến trúc máy tính song song

**a. Máy tính truy cập ngẫu nhiên song song PRAM**

- 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.

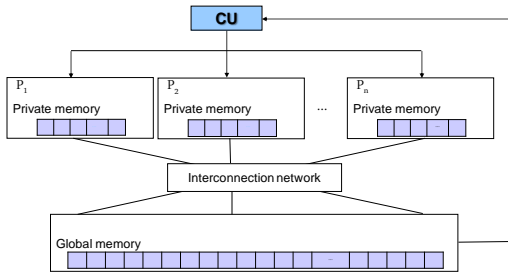
Các câu lệnh có thể bắt đầu thực hiện ở bất kỳ thời điểm nào, ở bất kỳ vị trí nào của bộ nhớ (riêng hoặc chung)

48



## 1.4 Kiến trúc máy tính song song

### a. Máy tính truy cập ngẫu nhiên song song PRAM



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

49

## 1.4 Kiến trúc máy tính song song

### 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.

50

## 1.4 Kiến trúc máy tính song song

### Một số điều cần lưu ý khi chuyển những thuật toán xây dựng cho MTSS tổng quát sang máy tính cụ thể

- ❑ Phải áp dụng một số các ràng buộc để đảm bảo chương trình thực hiện được trên những máy tính đó.
- ❑ Về hình thức, phải thực hiện một trong những điều kiện sau:
  - **EREW**: loại trừ vấn đề xung đột đọc/ghi ( $E_{exclusive} R_{read} + E_{exclusive} W_{write}$ )
  - **CREW**: cho phép đọc đồng thời, nhưng không cho phép xung đột khi ghi ( $C_{concurrent} R_{read} + E_{exclusive} W_{write}$ )
  - **CRCW**: Cho phép đọc, ghi đồng thời ( $C_{concurrent} R_{read} + C_{concurrent} W_{write}$ )

51

## 1.4 Kiến trúc máy tính song song

### b. Kiến trúc SIMD

#### Cấu trúc:

- Các phần tử xử lý (PE) đều được điều hành bởi một đơn vị điều khiển (CU).
- Các phần tử xử lý nhận được cùng một lệnh từ CU nhưng hoạt động trên những tập dữ liệu khác nhau.

#### Đặc tính:

- Phân tán việc xử lý trên nhiều phần cứng
- Thao tác đồng thời trên nhiều phần tử dữ liệu
- Thực hiện cùng một tính toán trên tất cả các phần tử dữ liệu.

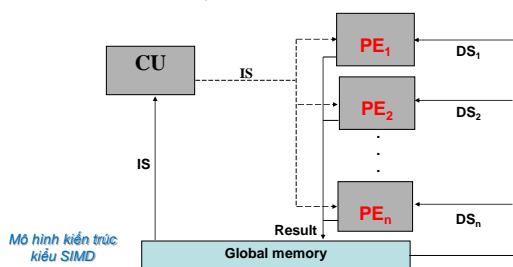
52

## 1.4 Kiến trúc máy tính song song

### b. Kiến trúc SIMD (tt)

IS: Instruction Stream  
LM: Local Memory

PE: Processing Element  
DS: Data Stream

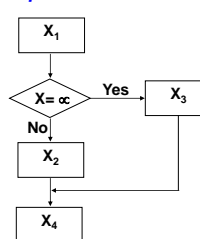


Mô hình kiến trúc kiểu SIMD

53

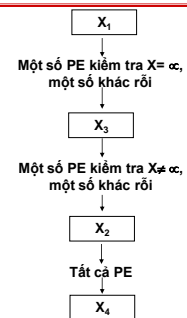
## 1.4 Kiến trúc máy tính song song

### Ví dụ



(a) thực hiện theo SISD,

Trong đó,  $X_1$ ,  $X_2$ ,  $X_3$ , và  $X_4$  là các khối của câu lệnh



(b) thực hiện theo SIMD

54

## 1.4 Kiến trúc máy tính song song

### b. Kiến trúc MISD

**BXL hình ống chính là BXL kiểu MISD**

**Nguyên lý hình ống (pipeline):** dựa trên nguyên tắc:

- Phân đoạn hoặc chia nhỏ một tiến trình thành một số tiến con để thực hiện trong các pha liên tiếp.
- Mỗi một giai đoạn của một tiến trình được thực hiện tuần tự. Sau khi thực hiện xong một pha thì bắt đầu thực hiện giai đoạn của tiến trình tiếp theo.
- Mỗi pha thực hiện xong sẽ truyền kết quả cho pha tiếp theo.

**Tóm lại, theo nguyên lý hình ống:**

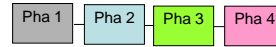
Khi một giai đoạn công việc đang thực hiện thì một giai đoạn khác có thể nạp dữ liệu vào, và dữ liệu vào của giai đoạn này có thể là kết quả của giai đoạn trước nó.

55

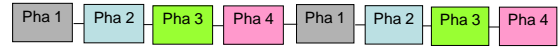
## 1.4 Kiến trúc máy tính song song

**Ví dụ: Thực hiện tuần tự và hình ống của 2 tiến trình gồm 4 giai đoạn**

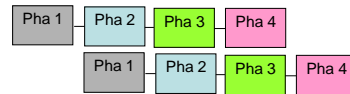
- Giả sử một tiến trình được chia thành 4 giai đoạn:



- Thực hiện tuần tự 2 tiến trình phải qua 8 giai đoạn:



- Thực hiện theo hình ống chỉ cần trải qua 5 giai đoạn:



Tổng thời gian tính toán tuần tự là:  $2 * (S_1 + S_2 + S_3 + S_4)$

Tổng thời gian tính toán hình ống là:  $S_1 + S_2 + S_3 + S_4 + S_4$

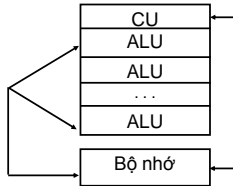
56

## 1.4 Kiến trúc máy tính song song

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.



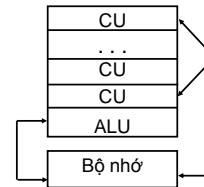
Xử lý hình ống theo ALU

57

## 1.4 Kiến trúc máy tính song song

### - Hình ống theo đơn vị câu lệnh:

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

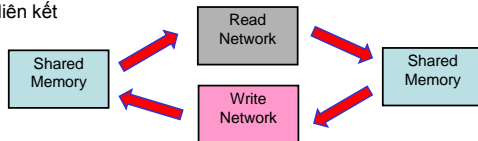


Xử lý hình ống theo CU

58

## 1.4 Kiến trúc máy tính song song

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



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:

**GD1:** Đọc dữ liệu: đọc dữ liệu từ bộ nhớ chia sẻ (Shared Memory).

**GD2:** Chuyển tải dữ liệu: chuyển dữ liệu từ bộ nhớ tới các phần tử xử lý PE thông qua mạng đọc (Read Network).

**GD3:** Thực hiện câu lệnh: sử dụng PE để thực hiện các câu lệnh.

**GD4:** Chuyển tải dữ liệu: chuyển các kết quả từ các PE tới bộ nhớ thông qua mạng ghi (Write Network).

**GD5:** Lưu trữ dữ liệu: ghi lại các kết quả vào bộ nhớ chia sẻ.

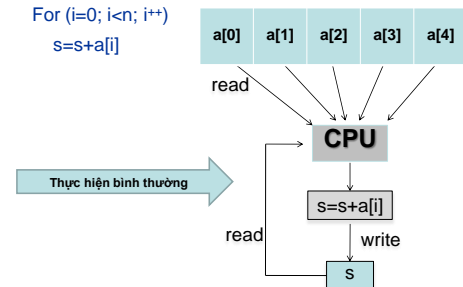
59

## 1.4 Kiến trúc máy tính song song

### Một ví dụ đơn giản

Tính tổng  $n$  phần tử của một mảng  $a$ :

```
For (i=0; i<n; i++)
    s=s+a[i]
```



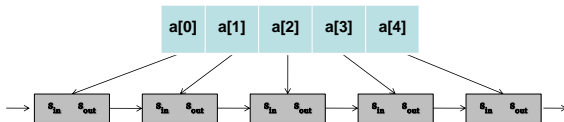
60

## 1.4 Kiến trúc máy tính song song

Tính tổng  $n$  phần tử của một mảng  $a$ :

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

Thực hiện theo đường ống



61

## 1.4 Kiến trúc máy tính song song

Ví dụ: Sắp xếp  $n$  phần tử bằng kỹ thuật đường ống

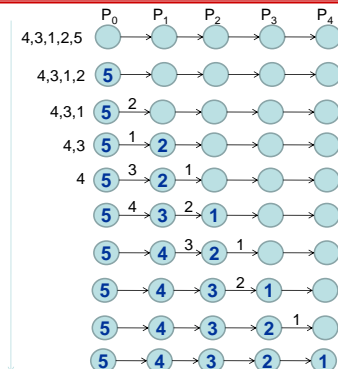
Ý tưởng thuật toán (insertion sort):

- $P_0$  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, Pi-1);
If (number > x) {
    send(&x, Pi-1);
    x = number;
} else send(number, Pi+1);
```

62

## 1.4 Kiến trúc máy tính song song



63

## 1.4 Kiến trúc máy tính song song

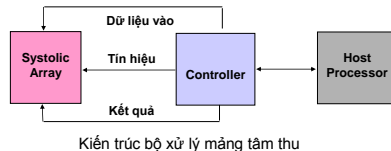
c. Bộ xử lý mảng tâm thu SAP (Systolic Array Processor)

- [Do Kung](#) và [Leiserson](#) đề xuất 1978
- Là một mảng các phần tử xử lý dữ liệu (DPU) được kết nối cục bộ với nhau.
- Mỗi DPU được xem như một tế bào, một ô trong mảng, bao gồm:
  - Một số thanh ghi (registers)
  - Một bộ cộng (adder)
  - Các mạch điều khiển
  - Đơn vị logic-số học ALU.

64

## 1.4 Kiến trúc máy tính song song

c. Bộ xử lý mảng tâm thu SAP (cont.)



Kiến trúc bộ xử lý mảng tâm thu

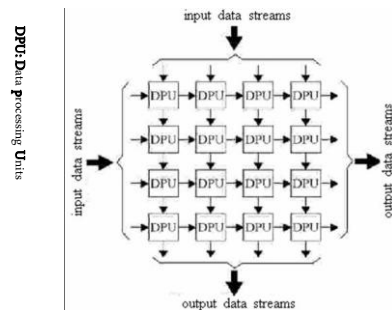
Trong kiến trúc SAP,

- 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

65

## 1.4 Kiến trúc máy tính song song

c. Bộ xử lý mảng tâm thu SAP (tt)

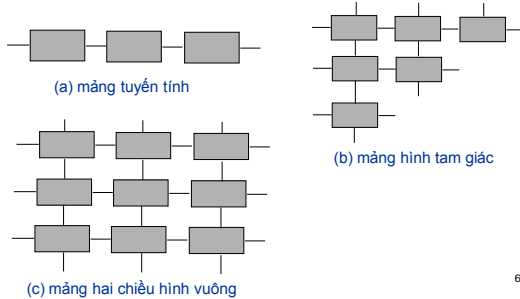


66

## 1.4 Kiến trúc máy tính song song

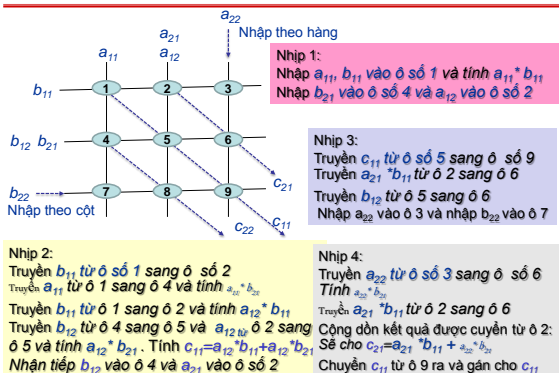
### c. Bộ xử lý mảng tâm thu SAP (cont.)

SA có thể tổ chức theo nhiều cấu hình topology khác nhau



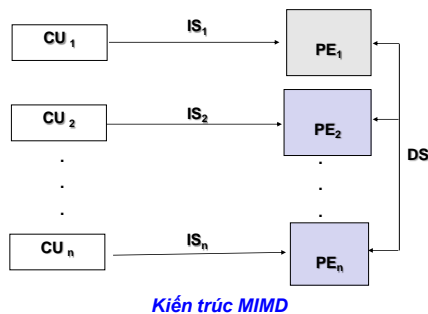
67

## 1.4 Kiến trúc máy tính song song



70

## 1.4 Kiến trúc máy tính song song



71

## 1.4 Kiến trúc máy tính song song

### c. Bộ xử lý mảng tâm thu SAP (tt)

Xét bài toán nhân 2 ma trận cỡ  $2 \times 2$ :  $A \times B = C$

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

$$c_{ij} = \sum_k a_{ik} * b_{kj}$$

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

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

Sử dụng bộ nhớ SAP hai chiều hình vuông để tính

68

## 1.4 Kiến trúc máy tính song song

### d. Kiến trúc MIMD

- Loại đa BXL hoặc còn gọi là hệ thống đa máy tính
- Trong đó mỗi BXL có đơn vị điều khiển (CU) riêng và thực hiện chương trình riêng của mình.

#### Những đặc trưng của MIMD:

- Xử lý phân tán trên một số BXL độc lập
- Chia sẻ với nhau một số tài nguyên, trong đó có hệ thống bộ nhớ.
- Mỗi BXL thao tác độc lập và có thể thực hiện đồng thời với nhau.
- Mỗi BXL có thể thực hiện một chương trình riêng.

## Câu hỏi chương 1

- Nêu đặc trưng cơ bản của kiến trúc máy tính tuần tự của VN
- Các kiến trúc máy tính có thể được phân loại như thế nào? dựa vào những yếu tố nào để phân loại?
- Một hệ thống như thế nào được gọi là máy tính song song?
- Máy tính kiểu MIMD khác với mạng các máy tính như thế nào?
- Nêu nguyên lý xử lý theo hình ống. Những bài toán có những tính chất gì thì thích hợp với kiến trúc xử lý hình ống?
- Cần bao nhiêu nhịp để thực hiện nhân hai ma trận  $100 \times 100$  trên SAP có  $100 \times 100$  phần tử xử lý? Nếu sử dụng hệ  $1000 \times 1000$  PE thì hệ nào tốt hơn (nhanh hơn)?
- Một công việc được chia thành  $m$  công việc con, mỗi công việc con đòi hỏi một đơn vị thời gian để thực hiện. Hỏi cần bao nhiêu đơn vị thời gian để hệ hình ống gồm  $m$ -bộ xử lý tuyến tính (gồm  $m$ -pha thực hiện) thực hiện được nhiệm vụ cho trước?

72

## CHƯƠNG 2. CÁC THÀNH PHẦN CỦA MT SONG SONG

## NỘI DUNG

## 2.1 Bộ nhớ của MTSS

## 2.2 Mạng kết nối các thành phần của MTSS

## 2.3 Chương trình dịch và hệ điều hành

## 2.4 Kết luận

73

## 2.1 Bộ nhớ của MTSS

## Một số vấn đề cần quan tâm trong kiến trúc MTSS

- *Sử dụng nhiều thanh ghi:*  
⇒ sẽ làm giảm hiệu ứng phụ của các ngắt
- *Sử dụng không gian nhớ lớn:*  
⇒ làm giảm hiệu ứng phụ của sự đổi chỗ khi thực hiện một lệnh.  
⇒ tăng hiệu quả trao đổi dữ liệu của hệ thống.
- *Xây dựng bộ lập lịch (Scheduling):* nhằm cấp phát hữu hiệu từng nhiệm vụ đơn lẻ cho các BXL cho một cách đồng bộ.
- *Đồng bộ các BXL (Synchronization):* điều khiển nhiều tiến trình hoạt động đồng thời, cùng truy cập đến một số hữu hạn các tài nguyên dùng chung, tránh được sự tắc nghẽn (deadlock)

74

## 2.1 Bộ nhớ của MTSS

## Một số vấn đề cần quan tâm trong kiến trúc MTSS (cont.)

▪ *Thiết kế cấu hình mạng:* tập trung vào việc kết nối giữa BXL với BXL, giữa BXL với bộ nhớ trong hệ thống. Cấu hình topology của mạng kết nối là vấn đề rất quan trọng trong thiết kế hệ thống song song.

▪ *Phân đoạn (Partitioning):* xác định mức độ song song trong các thuật toán để xác định được các luồng xử lý đồng thời. Sự phân đoạn có thể thể hiện ở nhiều mức khác nhau: mức lệnh, mức thủ tục, hoặc mức chương trình, v.v.

▪ *Đảm bảo tin cậy (Reliability):* nếu có một BXL nào đó không thực hiện được công việc đảm nhiệm thì một BXL khác sẽ được thay thế để đảm bảo trong mọi tình huống công việc chung vẫn được hoàn thành.

75

## 2.1 Bộ nhớ của MTSS

## Bộ nhớ của MTSS được chia thành nhiều mức

**Bộ nhớ mức 1:** mức cao nhất, có dung lượng nhỏ nhất, truy cập nhanh và đắt nhất, thường gắn chặt với mỗi BXL thành bộ nhớ cục bộ (local memory-khác với bộ nhớ chia sẻ).

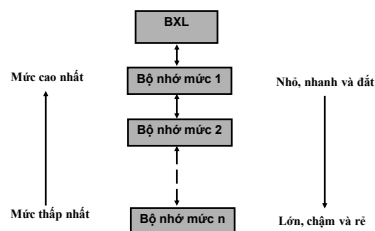
**Bộ nhớ mức 2:** Truy cập chậm hơn và rẻ hơn mức 1, v.v.

**Về nguyên tắc,** dữ liệu được chuyển đổi giữa các mức lân cận của các bộ nhớ và hoàn toàn được điều khiển bởi bộ nhớ mức 1.

**Về lý thuyết,** trong cấu trúc phân cấp bộ nhớ, tốc độ truy cập bộ nhớ trung bình gần bằng tốc độ truy cập ở mức cao nhất (mức 1), nhưng chi phí của các đơn vị nhớ trung bình lại gần với giá của bộ nhớ ở mức thấp nhất (mức n).

76

## 2.1 Bộ nhớ của MTSS



Phân cấp hệ thống bộ nhớ

77

## 2.1 Bộ nhớ của MTSS

## 2.1.1 Bộ nhớ kết hợp (AM – Associative Memory)

AM bao gồm các ô nhớ (cell)

Mỗi ô nhớ của AM có 4 đầu vào và hai 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

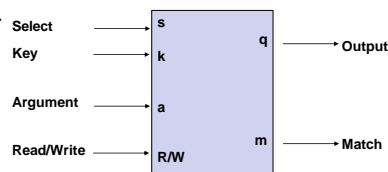
. Bit khoá  $k$

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

Hai kết quả ở đầu ra:

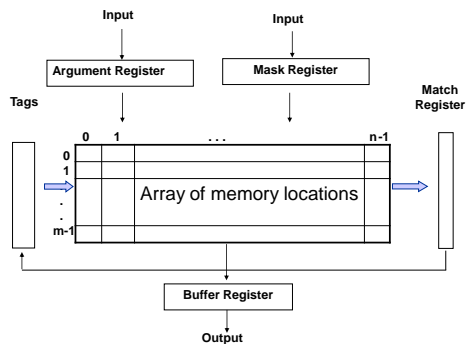
. Bit đối sánh  $m$  chỉ ra dữ liệu được lưu trong bộ nhớ có đối sánh được với bit đối số  $a$ .

. Bit kết quả ra  $q$ .



78

## 2.1 Bộ nhớ của MTSS - Cấu trúc của bộ nhớ kết hợp



79

## 2.1 Bộ nhớ của MTSS

**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.

80

## 2.1 Bộ nhớ của MTSS

### 2.1.2 Mô hình 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 độc lập.

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:

81

## 2.1 Bộ nhớ của MTSS

### Các phương thức truy cập bộ nhớ

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

82

## 2.1 Bộ nhớ của MTSS

- **Ghi đồng thời có ưu tiên (Priority CW)**: các BXL được gán 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 gán 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.

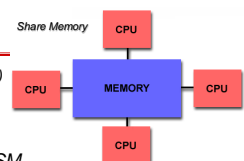
83

## 2.1 Bộ nhớ của MTSS

### 2.1.2 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)



84

## 2.1 Bộ nhớ của MTSS

### a. Mô hình UMA của bộ nhớ chia sẻ

#### Đặc điểm:

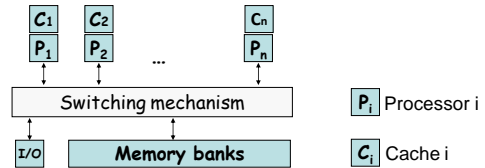
- Các BXL làm việc nhờ cơ chế chuyển mạch tập trung (*central switching mechanism*) để điều khiển việc truy cập tới bộ nhớ chia sẻ.
- Các BXL đều có thể truy cập đến bộ nhớ chia sẻ toàn cục (global shared memory)
- Thời gian truy cập vào bộ nhớ là như nhau đối với mọi BXL.
- Một BXL này có thể trao đổi 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.

85

## 2.1 Bộ nhớ của MTSS

### a. Mô hình UMA của bộ nhớ chia sẻ (tt)

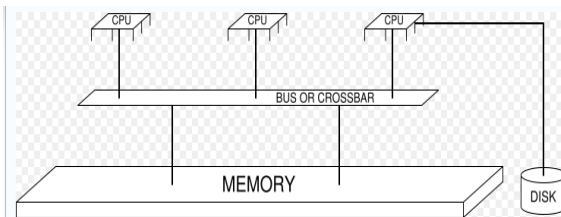
- Nếu tất cả các BXL của máy tính đều có thời gian truy cập đến các thiết bị là như nhau thì gọi là *máy tính đa bộ xử lý đối xứng* - *Symmetric MultiProcessor (SMP) machines*
- Máy tính với kiến trúc UMA còn được gọi: CC-UMA - *Cache Coherent UMA*



86

## 2.1 Bộ nhớ của MTSS

### a. Mô hình UMA của bộ nhớ chia sẻ (tt)



87

## 2.1 Bộ nhớ của MTSS

### b. Mô hình NUMA của bộ nhớ chia sẻ

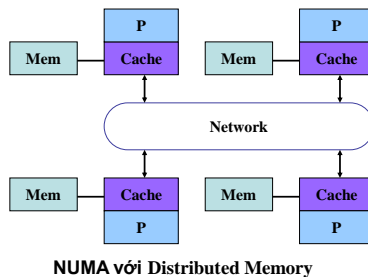
#### Đặc điểm

- Bộ nhớ chia sẻ được phân tán và chia thành các *modun* nhớ độc lập.
- Bộ nhớ chia sẻ được phân tán cho tất cả các BXL thành bộ nhớ cục bộ và tất cả các *modun* nhớ sẽ là bộ nhớ chung cho các BXL.
- Mô hình NUMA thường được tạo thành từ hai hoặc nhiều SMPs 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 *modun* 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

88

## 2.1 Bộ nhớ của MTSS

### b. Mô hình NUMA của bộ nhớ chia sẻ (tt)



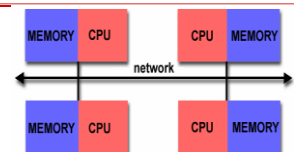
89

## 2.1 Bộ nhớ của MTSS

### 2.1.3. Bộ nhớ phân tán

#### Đặc điểm:

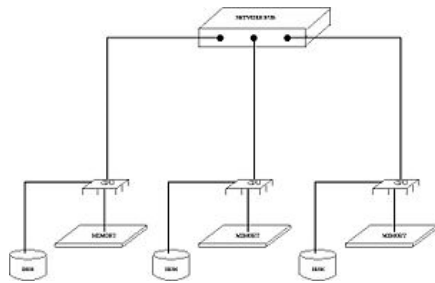
- Có một đường kết nối giữa các BXL
- Các BXL đều có bộ nhớ riêng.
- Địa chỉ bộ nhớ của một BXL nào đó không được biết bởi BXL khác. Do đó không có khái niệm không gian địa chỉ tổng thể (global address space) qua các BXL khác.
- Vì BXL này không thể tự do đọc/ghi vào bộ nhớ của một BXL khác nên khái niệm Cache coherent không áp dụng ở đây.
- Khi cần thiết người lập trình có nhiệm vụ chuyển dữ liệu từ ô nhớ của BXL này sang ô nhớ của một BXL khác.



90

## 2.1 Bộ nhớ của MTSS

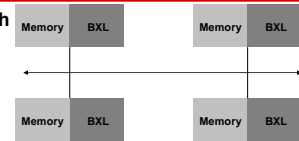
### 2.1.3. Bộ nhớ phân tán



91

## 2.1 Bộ nhớ của MTSS

### 2.1.3. Bộ nhớ đa máy tính



#### Đặc điểm:

- Mỗi nút trong hệ thống đa máy tính cũng chính là một máy tính có bộ nhớ riêng, không chia sẻ với những BXL khác.
- Các BXL trao đổi với nhau thông qua việc gửi và nhận các thông điệp (messages)
- Không tồn tại bộ nhớ chia sẻ chung mà mỗi BXL có bộ nhớ cục bộ riêng của chúng.
- Việc trao đổi dữ liệu trong mạng theo mô hình point to point thông qua sự liên kết tĩnh giữa các BXL.

92

## 2.1 Bộ nhớ của MTSS

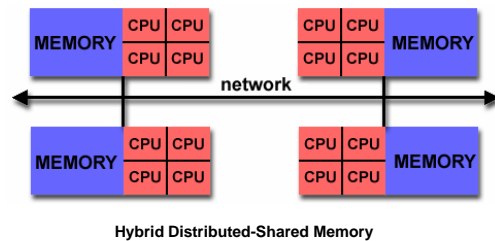
### 2.1.4 Máy tính với bộ nhớ lai (Hybrid Memory)

- Kết hợp giữa kiến trúc **chia sẻ bộ nhớ chung** và **bộ nhớ phân tán**.
- Hệ thống là nhiều cụm máy tính khác nhau. Mỗi cụm là các máy tính với bộ nhớ có kiến trúc chia sẻ và có bộ nhớ toàn cục riêng cho cụm đó.
- Các BXL trong một cụm chia sẻ bộ nhớ chung có thể truy cập đến bộ nhớ toàn cục riêng của cụm đó.
- Thành phần bộ nhớ phân tán được hiểu như là một cụm các bộ nhớ toàn cục của mỗi cụm.
- Các BXL chỉ có thể truy cập đến bộ nhớ toàn cục trong thành phần chia sẻ bộ nhớ phân tán của chúng, chứ không truy cập được bộ nhớ của các thành phần chia sẻ bộ nhớ chung khác.

93

## 2.1 Bộ nhớ của MTSS

### 6. Máy tính với bộ nhớ lai



Hybrid Distributed-Shared Memory

94

## 2.1 Bộ nhớ của MTSS

### Tóm lại,

Hai điều cần phải quan tâm trong thiết kế bộ nhớ của hệ thống song song. Đó là,

- **Băng thông** (bandwidth): đường kết nối các BXL, bộ nhớ
- **Độ trễ bộ nhớ** (memory latency), độ trễ bộ nhớ được hiểu như là khoảng thời gian từ khi **BXL yêu cầu dữ liệu từ bộ nhớ** đến khi **BXL nhận được dữ liệu tương ứng**.

Trong hệ thống máy tính song song băng thông phải đủ rộng để thực hiện việc truyền thông của các BXL.

Khi các mô đun nhớ được chia sẻ thì việc cạnh tranh sử dụng không gian nhớ phải được cực tiểu hóa. Do đó, độ trễ bộ nhớ cũng phải được cực tiểu hóa.

95

## 2.2 Mạng kết nối các thành phần của MTSS

### Một vài nhận xét:

- Trong hầu hết các kiến trúc song song như: những hệ đa BXL chia sẻ bộ nhớ, đa bộ xử lý đa bộ nhớ, v.v. thì vấn đề quan trọng nhất trong thiết kế là xác định sự kết nối giữa các BXL và bộ nhớ với nhau.
- Một kiến trúc lý tưởng là kiến trúc trong đó, mỗi BXL đều kết nối được với các BXL còn lại. Khi đó nó tạo ra một đồ thị đầy đủ.
- Ví dụ, nếu hệ có p BXL thì sẽ có  $p(p-1)/2$  đường liên kết. Để nhận thấy kiến trúc loại này sẽ rất phức tạp, nhất là khi p đủ lớn.

96



## 2.2 Mạng kết nối các thành phần của MTSS

• Có hai loại cấu hình topology cho mạng liên kết: liên kết tĩnh và liên kết động.

➤ **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.

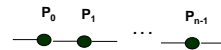
97

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.1 Liên kết tuyến tính và vòng (linear and ring)

#### a. Liên kết tuyến tính

Các BXL được liên kết với nhau theo dãy và được đánh số theo thứ tự tăng dần



#### Nhận xét:

• Trong mạng liên kết tuyến tính, trừ hai phần tử đầu và cuối, tất cả các BXL bên trong đều có hai láng giềng là BXL trước và sau nó.

• Đây là dạng liên kết đơn giản, nhưng dữ liệu cũng cần phải chuyển qua nhiều BXL, do đó sự truyền thông dữ liệu giữa các BXL, đặc biệt là giữa BXL đầu và cuối sẽ bị chậm lại khi số BXL khá lớn.

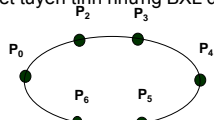
98

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.1 Liên kết tuyến tính và vòng

#### b. Liên kết vòng

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



#### Nhận xét:

• Trong mạng liên kết vòng, 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**.

• Tất nhiên sự truyền thông trong mạng liên kết vòng, nhất là giữa những BXL ở xa nhau thì cũng vẫn bị trễ.

99

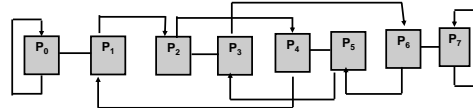
## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.2 Mạng liên kết xáo trộn hoàn hảo (Perfect Shuffle Exchange)

Giả sử có N BXL:  $P_0, P_1, \dots, P_{N-1}$ , với N là lũy thừa của 2.

Trong **mạng liên kết xáo trộn hoàn hảo**, đường liên kết một chiều từ  $P_i$  tới  $P_j$  được xác định như sau:

$$j = \begin{cases} 2i & \text{với } 0 \leq i \leq N/2 - 1 \\ 2i+1-N & \text{với } N/2 \leq i \leq N-1 \end{cases}$$

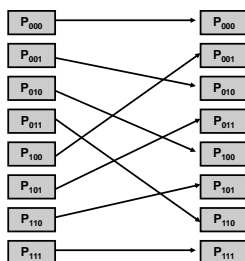


Mạng liên kết xáo trộn hoàn hảo một chiều với  $N = 8=2^3$ , trong đó sự liên kết xáo trộn được ký hiệu bằng mũi tên.

100

## 2.2 Mạng kết nối các thành phần của MTSS

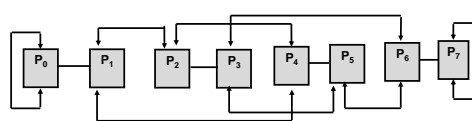
Một biểu diễn khác của mạng liên kết xáo trộn hoàn hảo



101

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.2 Mạng liên kết xáo trộn hoàn hảo hai chiều



Mạng liên kết xáo trộn hoàn hảo hai chiều với  $N = 8$ , trong đó sự liên kết xáo trộn được ký hiệu bằng mũi tên hai chiều.

102

## 2.2 Mạng kết nối các thành phần của MTSS

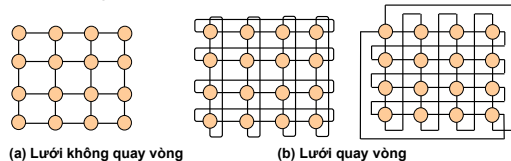
### 2.2.3 Mạng liên kết lưới hai chiều (two-dimensional mesh)

#### Đặc điểm:

Mỗi BXL được liên kết với bốn láng giềng: trên, dưới, bên trái và bên phải.

Mạng liên kết lưới hai chiều được sử dụng để thiết kế các máy tính ILLIAC IV, MPP (Massively Parallel Processor), DAP (ICL Distributed Array Processor), v.v.

Có hai dạng liên kết lưới:



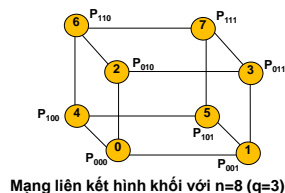
103

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.4 Mạng liên kết siêu khối hoặc hình khối n-chiều

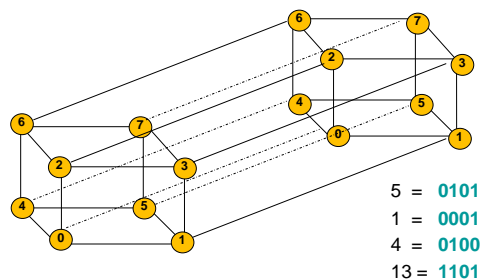
Giả sử có  $n$  bộ xử lý:  $P_0, P_1, \dots, P_{n-1}$ , với  $n = 2^q$ ,  $q \geq 0$ . Nếu mỗi BXL cần liên kết với đúng  $q$  bộ xử lý lân cận thì có thể dùng mạng liên kết theo hình siêu khối  $n$  chiều.

Trong mạng liên kết hình khối các chỉ số của các BXL được đánh nhị phân, hai BXL được gọi là láng giềng với nhau nếu nhận chỉ số của chúng sai khác nhau đúng một bit.



104

## 2.2 Mạng kết nối các thành phần của MTSS



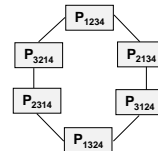
105

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.5 Mạng liên kết hình sao (star)

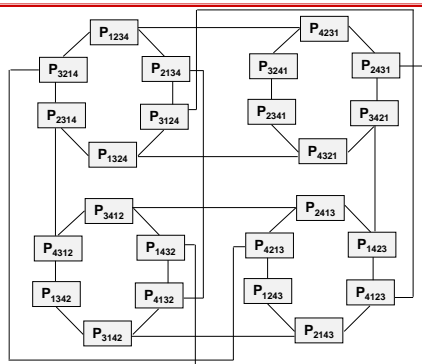
- Ký hiệu là  $S_n$
- Có  $n!$  bộ xử lý
- Nhãn của mỗi BXL sẽ tương ứng với một hoán vị của  $n$  ký hiệu.
- BXL  $P_i$  được kết nối với BXL  $P_j \Leftrightarrow j$  nhận được từ  $i$  bằng cách thay ký hiệu thứ  $k$  của hoán vị  $n$  phần tử  $1, \dots, n$ , với  $2 \leq k \leq n$ .

Ví dụ, trong  $S_4$  ( $n = 4$ ) có  $4! = 24$  BXL. Hai BXL  $P_{2134}$  và  $P_{3124}$  là có liên kết với nhau vì 3124 nhận được từ 2134 bằng cách đổi chỗ vị trí đầu (số 2) với vị trí thứ ba (số 3). Tương tự suy ra  $P_{2134}$  và  $P_{4132}$  là có liên kết với nhau.



106

### Mạng liên kết hình sao với 24 bộ xử lý



107

## 2.2 Mạng kết nối các thành phần của MTSS

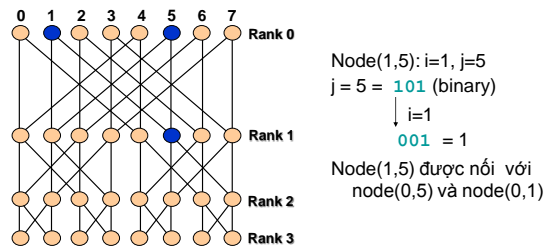
### 2.2.5 Mạng liên kết Butterfly

- $(k+1)2^k$  nút được chia thành  $(k+1)$  ranks, mỗi rank chứa  $n=2^k$  nút.
- Các ranks được đánh số từ 0 đến  $k$
- Ký hiệu  $\text{Node}(i,j)$ : nút thứ  $j$  trên rank thứ  $i$
- $\text{Node}(i,j)$  được nối với 2 nút trên rank  $i-1$ :  $\text{node}(i-1,j)$  và  $\text{node}(i-1,m)$ , ở đây  $m$  là số nguyên có được bằng cách đảo bit thứ  $i$  trong xâu nhị phân biểu diễn  $j$
- Nếu  $\text{node}(i,j)$  được nối đến  $\text{node}(i-1,m)$ , thì  $\text{node}(i,m)$  được nối với  $\text{node}(i-1,j)$

108

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.5 Mạng liên kết Butterfly (tt)



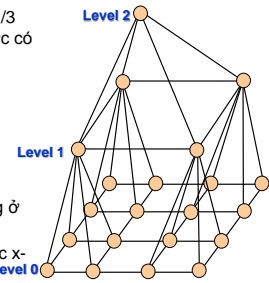
Mạng liên kết Butterfly với 4 ranks và 32 BXL

109

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.6 Mạng liên kết Pyramid

- Một tháp chiều cao  $d$  gồm  $(4^{d+1}-1)/3$  BXL được phân tán trong  $d+1$  mức có các tính chất sau:
  - Có  $4^{d-2}$  BXL ở mức  $d$ .
  - Có  $4^{d-1}$  BXL ở mức  $d-1$ , và
  - Có  $4^d$  BXL ở mức  $d-2$
- Chỉ có 01 BXL ở mức  $d$
- Đối với mức  $x$ :
  - Được nối với 04 nút láng giềng ở cùng một mức nếu  $x < d$
  - Được nối với 04 nút con ở mức  $x-1$  nếu  $x \geq 1$
  - Được nối với 04 nút con ở mức  $x+1$  nếu  $x \leq d-1$



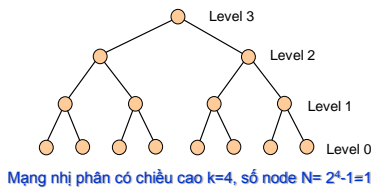
Mạng liên kết Pyramid  
 Có  $d=2$  và  $(4^{d+1}-1)/3 = 21$  BXL

110

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.7 Mạng liên kết nhị phân (binary tree)

- Chiều cao của cây gồm  $k$  mức, được đánh số từ 0 đến  $k-1$
- Mỗi BXL là một node của cây, có  $N=2^k-1$  nodes
- Mỗi BXL ở mức  $i$  được nối với nút cha ở mức  $i+1$  và 2 nút con ở mức  $i-1$



Mạng nhị phân có chiều cao  $k=4$ , số node  $N=2^4-1=15$

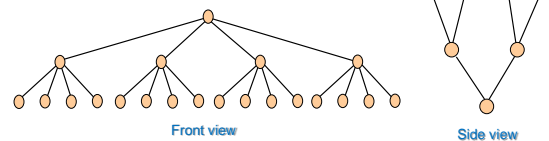
111

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.8 Mạng Hypertree

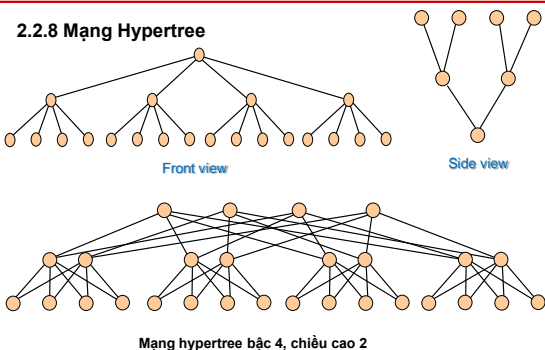
#### Đặc điểm:

- $d$  là chiều cao cây được đánh số từ 0 đến  $d-1$  với  $4^d$  nút lá
- Có  $2^d(2^{d-1}-1)$  nodes
- Nhìn từ phía trước mạng như một cây có chiều cao  $d$
- Nhìn từ phía bên mạng như một cây nhị phân đảo ngược có chiều cao  $d$



## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.8 Mạng Hypertree



Mạng hypertree bậc 4, chiều cao 2

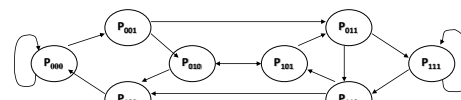
## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.9 Mạng liên kết De Bruijn

Gồm  $N = d^k$  BXL

Chỉ số mỗi BXL biểu diễn với  $k$  chữ số  $(a_{k-1}a_{k-2}...a_1a_0)$ , với  $a_j \in \{0, 1, ..., d-1\}$ ,  $j = 0, 1, ..., k-1$ .

số bộ xử lý có thể đạt được bởi  $(a_{k-1}a_{k-2}...a_1a_0)$  là  $(a_{k-2}a_{k-3}...a_0q)$  và  $(qa_{k-1}a_{k-2}...a_2a_1)$ ,  $q = 0, 1, ..., d-1$ .



Ví dụ một mạng De Bruijn với  $d = 2$ , và  $k = 3$ ,  $N = 2^3$ , có ba chữ số  $a_2, a_1, a_0$

114

## 2.2 Mạng kết nối các thành phần của MTSS

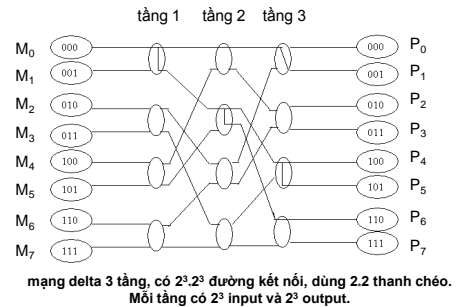
### 2.2.10 Mạng liên kết Delta

- Có  $k$  tầng.
- Mỗi tầng gồm các thanh chéo có  $m$  input và  $n$  output. tổng cộng  $m \cdot n$  thanh chéo.
- Mạng bao gồm  $m^k$  input và  $n^k$  output; do đó nó mạng có  $m^k \cdot n^k$  chuyển mạch.
- Các kết nối chuyển mạch cho phép tạo một đường dẫn chính xác từ các input đến các output.
- Nếu  $A$  là địa chỉ đích của một kết nối mong muốn trên cơ sở của  $n$ , thì các số của  $A$  biểu diễn các thanh chéo đặt ra để thiết lập kết nối theo dự kiến đó.

115

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.10 Mạng liên kết Delta



116

## 2.2 Mạng kết nối các thành phần của MTSS

### 2.2.8 Một số mạng kết nối khác

Tham khảo:

- [6] M. Sasikumar, Dinesh Shikhar, P. Ravi Prakash, *Introduction to Parallel Processing*, Prentice – Hall, 2002  
 [7] Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation*, Springer, 2006.  
 [8] Michael J. Quinn, *Parallel Computing Theory and Practice*, McGraw-Hill, 2004

117

## 2.4 Kết luận

### Vấn đề trọng tâm của chương 2

1. Tìm hiểu các tài nguyên và mối quan hệ của chúng trong hệ thống để tận dụng được hết khả năng xử lý song song.
2. Các bộ nhớ được tổ chức thành bộ nhớ kết hợp, bộ nhớ truy cập ngẫu nhiên, bộ nhớ chia sẻ, v.v. là các mô hình chính cho việc thiết kế bộ nhớ MTSS.
3. Vấn đề quan trọng trong thiết kế kiến trúc của MTSS là xác định cách để kết nối các bộ xử lý với nhau sao cho hiệu quả nhất.
4. Các bộ xử lý có thể kết nối theo mạng liên kết tĩnh hoặc liên kết động.
5. Khảo sát một số cấu hình topology của mạng liên kết các bộ xử lý như: liên kết tuyến tính, liên kết xoắn trộn, liên kết lưới, liên kết hình sao và liên kết hình khối...

118

## 2.3 Chương trình dịch và các hệ điều hành

### Câu hỏi cuối chương

1. Nêu những vấn đề cần quan tâm khi thiết kế kiến trúc máy tính song song
2. Bộ nhớ kết hợp là gì? nêu nguyên lý hoạt động của bộ nhớ kết hợp.
3. Tại sao mạng liên kết lại đóng vai trò quan trọng trong kiến trúc MTSS?
4. Dựa vào định nghĩa chung của mạng liên kết hình khối để xây dựng cấu trúc topology cho mạng liên kết hình khối cho 16 bộ xử lý.
5. Nêu những đặc trưng cơ bản của chương trình dịch song song?
6. Nếu mục đích chính của hệ điều hành cho máy tính song song?
7. Xây dựng mạng liên kết theo mô hình xoắn trộn hoàn hảo cho 16 phần tử.

119

## Tiểu luận

### 1. Kiến trúc kết nối mạng

De Bruijn, Mạng cây nhị phân, Mạng Delta, Mạng Butterfly, Mạng Omega, Mạng Pyramid ([7] Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation*, Springer, page 72-80)

### 2. Chương trình dịch song song ([7] Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation*, Springer, page 83-90)

### 3. Hệ điều hành đa xử lý ([7] Seyed H. Roosta, *Parallel Processing and Parallel Algorithms, Theory and Computation*, Springer, page 91-100)

### 4. Tìm hiểu hệ điều hành Linux-Windows (liên quan đến đa luồng đa tiến trình)

120

## CHƯƠNG 3. GIỚI THIỆU VỀ LẬP TRÌNH SONG SONG

### NỘI DUNG

1. Những khái niệm cơ sở của lập trình song song
2. Các ngôn ngữ lập trình song song
3. Các loại phụ thuộc dữ liệu trong chương trình
4. Phương pháp biến đổi chương trình tuần tự sang chương trình song song

121

### 3.1 Giới thiệu chung

Trong *môi trường song song*:

- Các câu lệnh của chương trình có thể thực hiện đan xen lẫn nhau.
- Ở cùng một thời điểm có thể có nhiều hơn một lệnh được thực hiện, nghĩa là mỗi chương trình sẽ tự chủ thực hiện các tiến trình của mình.
- Các chương trình phải tương tác với nhau và việc thực hiện của chúng ảnh hưởng tới nhịp độ thực hiện của nhau.
- Trong LTSS, người lập trình không chỉ viết chương trình, dữ liệu như trong môi trường tuần tự mà còn phải sử dụng các công cụ để đồng bộ hoá (synchronize) và điều khiển sự tương tác giữa các tiến trình.
- Người lập trình cần tạo ra và lập lịch cho các tiến trình, nghĩa là sự thực hiện chương trình có thể nhìn thấy được bởi người lập trình.

123

### 3.1 Giới thiệu chung

#### Sự khác nhau cơ bản giữa LT tuần tự và LT song song

Trong *môi trường lập trình tuần tự*:

- Các câu lệnh được thực hiện tuần tự.
- Mỗi chương trình thực hiện sẽ tạo ra những tiến trình bên trong hệ thống mà người lập trình không quan sát được.
- Mỗi câu lệnh thực hiện không gây trở ngại cho các câu lệnh khác trong chương trình.

122

### 3.1 Giới thiệu chung

#### Vấn đề quan trọng trong LTSS là phải tận dụng được khả năng của các bộ xử lý.

Có hai cách tiếp cận để tận dụng khai thác các bộ xử lý:

1. Phát triển những ngôn ngữ lập trình cho phép thể hiện được việc thực hiện song song ở mức thuật toán, ví dụ như Fortran, C, v.v.
2. Xây dựng những chương trình dịch đủ mạnh để nhận dạng được các phân đoạn chương trình có thể thực hiện song song hay tuần tự.

Hai cách tiếp cận trên là bổ sung cho nhau, nếu chỉ áp dụng một cách thì không hiệu quả.

124

### 3.1 Giới thiệu chung

Một số phương pháp tiếp cận trong lập trình song song:

- *Lập trình song song kiểu SIMD với bộ nhớ chia sẻ*, trong đó truy cập bộ nhớ là đồng bộ (Synchronous).
- *Lập trình song song kiểu MIMD với bộ nhớ chia sẻ*, trong đó truy cập bộ nhớ là dị bộ (Asynchronous).
- *Lập trình song song kiểu MIMD với bộ nhớ phân tán*, trong đó truy cập bộ nhớ là dị bộ.
- *Lập trình song song kiểu SPMD với bộ nhớ chia sẻ*, trong đó truy cập bộ nhớ là dị bộ.
- *Lập trình song song kiểu SPMD với bộ nhớ phân tán*, trong đó truy cập bộ nhớ là dị bộ.
- *Lập trình song song kiểu MPMD với bộ nhớ chia sẻ*, trong đó truy cập bộ nhớ là dị bộ.

125

### 3.2 Các ngôn ngữ lập trình song song

#### Các yêu cầu đối với một NNLT song song

Ngoài các yêu cầu đối với một NNLT tuần tự, một NNLT song song cần phải có các chức năng sau:

1. Cung cấp cho người lập trình những cơ chế để khởi tạo, đồng bộ và trao đổi giữa các tiến trình.
2. Tạo ra được những chương trình độc lập với máy tính.
3. Phải hỗ trợ để tạo ra được những chương trình dễ đọc, dễ viết, dễ chuyển đổi, v.v.
4. Mô hình hoá được việc thực hiện song song.
5. Có khả năng điều chỉnh các tình huống mà ở đó các tiến trình đòi hỏi phải trao đổi, tương tác với nhau.

126

### 3.2 Các ngôn ngữ lập trình song song

#### Các tình huống thường gặp trong LT song song

1. Tại một thời điểm có một số tiến trình muốn truy cập vào một tài nguyên chung hoặc cập nhật vào một biến chia sẻ. Mà những tài nguyên đó chỉ cho phép một tiến trình truy cập tại mỗi thời điểm.
2. Khi một tiến trình được quyền truy cập vào tài nguyên chung thì nó sử dụng tài nguyên đó nhưng không được ngăn cản hoạt động của những tiến trình khác.
3. Khi một số tiến trình cùng kết hợp để thực hiện một số phép toán trên cơ sở quan sát hành động của nhau thì người lập trình phải lập lịch cho những tiến trình đó.

Có hai cách để giải quyết các tình huống trên:

127

### 3.2 Các ngôn ngữ lập trình song song

Tổng quát, có hai cách phát triển NNLT song song:

1. Mở rộng những ngôn ngữ lập trình tuần tự hiện có, bổ sung thêm những cấu trúc mới để thực hiện được song song và giải quyết được sự xung đột trong truy cập dữ liệu.
2. Xây dựng một ngôn ngữ lập trình song song mới.

129

### 3.2 Các ngôn ngữ lập trình song song

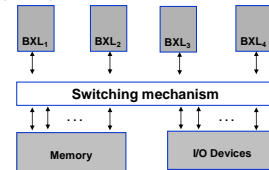
1. Tất cả các tiến trình phải sử dụng cấu trúc dữ liệu chung để tiện trong việc cập nhật và trao đổi với nhau.
2. Tất cả các tiến trình thực hiện sự đồng bộ bằng cách sử dụng hai hàm **wait()** và **pass()**. Khi trao đổi với nhau, một tiến trình chờ để nhận được một tín hiệu của tiến trình đối tác và khi nhận được thì hai tiến trình đó trao đổi trực tiếp với nhau.

128

### 3.2 Các ngôn ngữ lập trình song song

#### 3.2.1 Ví dụ minh họa

a. Cài đặt song song trên kiến trúc UMA với 4 BXL:



**Bài toán:** Xác định phương sai (variance) của một mẫu được cho bởi một danh sách các số thực  $r_1, r_2, \dots, r_n$ . Nghĩa là phải tính:

$$\sum_{i=1}^n \frac{(r_i - m)^2}{n} \quad \text{trong đó} \quad m = \sum_{i=1}^n \frac{r_i}{n}$$

130

### 3.2 Các ngôn ngữ lập trình song song

#### Cài đặt song song:

Các số thực  $r_i$  được lưu trữ trong bộ nhớ chia sẻ.

Bốn biến sau đây cũng được tạo ra trong bộ nhớ chia sẻ:

- **Sum** (lưu tổng giá trị):  $S = \sum_{i=1}^n r_i$
- **Mean** (lưu giá trị trung bình):  $m = \sum_{i=1}^n \frac{r_i}{n}$
- **Sum of square** (lưu tổng bình phương)  $SS = \sum_{i=1}^n (r_i - m)^2$
- **Variance** (lưu giá trị phương sai)  $\frac{SS}{n} = \frac{\sum_{i=1}^n (r_i - m)^2}{n}$

Quá trình thực hiện song song được mô tả như sau:

131

$$\sum_{i=1}^n \frac{(r_i - m)^2}{n} \quad m = \sum_{i=1}^n \frac{r_i}{n} \quad n=12; (r_i)=(4,2,1,3,3,2,2,4,5,1,4,5)$$

Shared memory					
Values				Process 1	Temporary
4	2	1	3	7	
3	2	2	4	Process 2	Temporary
5	1	4	5	8	
Sum	Sum of			Process 3	Temporary
36	Mean Squares			11	
	Variance			Process 4	Temporary
				10	

(a): Mỗi tiến trình cộng 3 giá trị được phân chia và lưu vào biến tạm thời của nó. Sau khi các tiến trình đã thực hiện xong việc tính các tổng con, sẽ có một tiến trình cộng dồn các giá trị này lại và lưu vào biến Sum ở bộ nhớ chia sẻ.

Shared memory				
Values				
4	2	1	3	3
2	2	4	5	1
4	5	1	4	5
Sum of				
Sum	Mean	Squares	Variance	
36	3	0		

Process 1	Temporary	7
Process 2	Temporary	8
Process 3	Temporary	11
Process 4	Temporary	10

(b): Một tiến trình đơn sẽ tính giá trị trung bình và lưu vào biến bộ nhớ chia sẻ Mean.

132

$$\sum_{i=1}^n \frac{(r_i - m)^2}{n} \quad m = \sum_{i=1}^n \frac{r_i}{n} \quad n=12; (r_i)=(4,2,1,3,3,2,2,4,5,1,4,5)$$

Shared memory

Values				
4	2	1	3	3
2	2	4	5	1
4	5	1	4	5
Sum of				
Sum	Mean	Squares	Variance	
36	3	22		

Process 1	Temporary	6
Process 2	Temporary	1
Process 3	Temporary	6
Process 4	Temporary	9

(c): Mỗi tiến trình phải xác định:  $(x_i - \text{Mean})^2$ ,  $\sum (x_i - \text{Mean})^2$  lưu vào biến tạm thời và cuối cùng cộng dồn để lưu vào biến của bộ nhớ chia sẻ.

Shared memory

Values				
4	2	1	3	3
2	2	4	5	1
4	5	1	4	5
Sum of				
Sum	Mean	Squares	Variance	
36	3	22	1.83	

Process 1	Temporary	6
Process 2	Temporary	1
Process 3	Temporary	6
Process 4	Temporary	9

(d): Một tiến trình đơn xác định variance bằng cách chia giá trị của Sum of Squares cho 12.

133

### 3.2 Các ngôn ngữ lập trình song song

#### 3.2.1 Ví dụ minh họa (tiếp)

b. Cài đặt song song trên NUMA với kiến trúc siêu khối 4 BXL:

**Bài toán:** Xác định phương sai (variance) của một mẫu được cho bởi một danh sách các số thực  $r_1, r_2, \dots, r_n$ . Nghĩa là phải tính:

$$\sum_{i=1}^n \frac{(r_i - m)^2}{n} \quad \text{trong đó} \quad m = \sum_{i=1}^n \frac{r_i}{n}$$

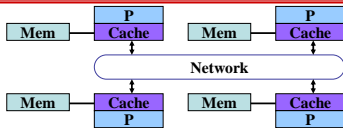
#### Cài đặt song song

- Kiến trúc này không có bộ nhớ chia sẻ chung;  $n$  giá trị  $r_1, r_2, \dots, r_n$  được phân phối ở các bộ nhớ cục bộ của mỗi BXL.
- Mỗi nút có 4 biến: 2 biến để lưu trữ các giá trị cộng dồn là **Sum** và **Sum of Squares**; một biến để lưu giá trị trung bình **Mean** và biến còn lại để lưu trữ giá trị phương sai **variance**.
- Quá trình tiếp tục thực hiện được mô tả như hình dưới đây.

134

### 3.2 Các ngôn ngữ lập trình song song

**Nhắc lại,**

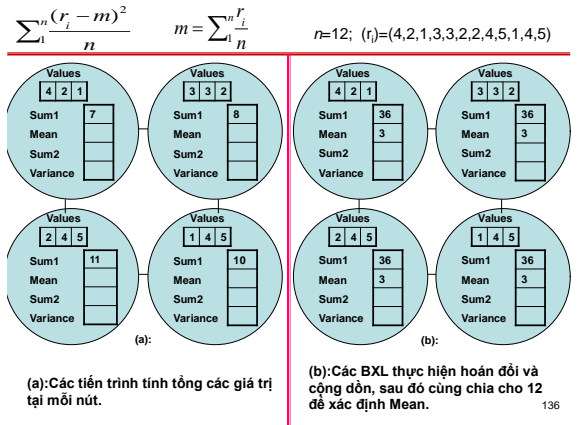


NUMA với Distributed Memory

#### Đặc điểm của NUMA:

- Có một đường kết nối các BXL lại với nhau.
- Bộ nhớ chia sẻ được phân tán cho tất cả các BXL thành bộ nhớ cục bộ và tất cả các *modun* nhớ sẽ là bộ nhớ chung cho các BXL.
- Các BXL được phép truy cập đồng thời tới một hay nhiều mô đun nhớ và có thể hoạt động độc lập với nhau.

135



136

$$\sum_{i=1}^n \frac{(r_i - m)^2}{n} \quad m = \sum_{i=1}^n \frac{r_i}{n} \quad n=12; (r_i)=(4,2,1,3,3,2,2,4,5,1,4,5)$$

(c): Mỗi tiến trình, với các giá trị riêng hiện có tại nút của mình sẽ thực hiện tính  $\sum (x_i - \text{Mean})^2$  và lưu vào biến Sum2.

Values				
4	2	1	3	3
2	2	4	5	1
4	5	1	4	5
Sum of				
Sum1	Mean	Sum2	Variance	
36	3	6		

(d): Các BXL thực hiện hoán đổi và cộng dồn, sau đó cùng chia cho 12 để xác định Variance.

Values				
4	2	1	3	3
2	2	4	5	1
4	5	1	4	5
Sum of				
Sum1	Mean	Sum2	Variance	
36	3	22	1.83	

137

### 3.2 Các ngôn ngữ lập trình song song

#### Giới thiệu một số ngôn ngữ lập trình song song

##### 3.2.2 Fortran 90

- Fortran 90** là ngôn ngữ lập trình chuẩn [ANSI](#).
- Fortran 90** là ngôn ngữ lập trình song song theo dữ liệu được nâng cấp từ Fortran 77 bằng cách bổ sung thêm một số đặc tính như:
  - Kiểu dữ liệu do User định nghĩa
  - Phép toán về Array, con trỏ
  - Các BXL hỗ trợ việc sử dụng các kiểu dữ liệu Short Integer, [Packed logical](#),...
  - Cấp phát bộ nhớ động.

138

### 3.2 Các ngôn ngữ lập trình song song

Người lập trình Fortran 90 dựa vào mô hình song song tương tự như PRAM gồm có những thành phần:

- Một CU (đơn vị điều khiển)
  - Một đơn vị xử lý logic, số học ALU
  - Bộ nhớ chia sẻ.
  - Một tập các BXL
- CPU thực hiện các câu lệnh tuần tự bằng cách truy cập vào các biến được lưu trữ trong bộ nhớ chia sẻ.
  - Đơn vị vector lưu trữ, đọc, ghi dữ liệu vào bộ nhớ chia sẻ và được CPU điều khiển để thực hiện song song.

139

### 3.2 Các ngôn ngữ lập trình song song - Fortran 90

**Fortran 90** có các kiểu dữ liệu chuẩn:

REAL, INTEGER, CHARACTER, LOGICAL, v.v.

Dạng khai báo tổng quát:

**Type** [(**kind**)] [, **attribute**] ... :: **Variable-List**

Trong đó,

- **Type** là kiểu cơ sở hoặc kiểu được định nghĩa bởi NSD
- **kind** là phần tùy chọn cùng một số kiểu được sử dụng để định nghĩa về kiểu cụ thể

Ví dụ: **CHARACTER (LEN = 10) :: s1**

Định nghĩa biến **s1** kiểu **CHARACTER** có thể chứa 10 ký tự.

- [, **attribute**] là danh sách các thuộc tính của Fortran được sử dụng để định nghĩa các đặc tính riêng của danh sách các biến.

+: phân tử phân cách giữa phần mô tả kiểu và danh sách các biến.

- **Variable-List**: danh sách các biến

140

### 3.2 Các ngôn ngữ lập trình song song - Fortran 90

Ví dụ về khai báo về mảng:

**INTEGER, DIMENSION (1 : 10) :: SA**

{Biến mảng SA có 10 phần tử kiểu số nguyên.}

+ Khai báo một hằng mảng:

**(/ 2,4,6,8,10 /)** hoặc **(/ (I, I = 2, 10, 2) /)**

+ Fortran 90 cho phép áp dụng các phép toán số học cho các biến mảng, nghĩa là toán hạng của các phép toán số học (+, -, \*, /) có thể là biến đơn hoặc biến mảng.

Ví dụ về phép toán trên biến mảng:

**INTEGER A(10), B(10), C**

**DO I = 1, 10, 1**

**A(I) = B(I) + C**

**END DO**

Ta có thể viết ngắn gọn hơn.

**A = B + C**

141

### 3.2 Các ngôn ngữ lập trình song song - Fortran 90

**Cấu trúc chương trình Fortran**

**PROGRAM** <tên chương trình>

**IMPLICIT NONE**

[phần đặc tả]

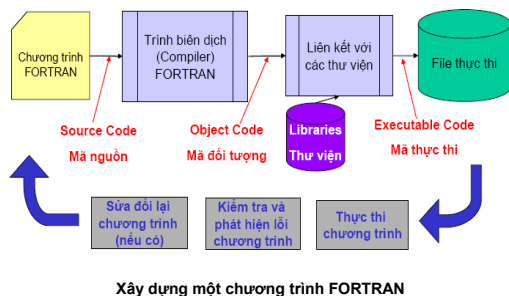
[phần thực hiện]

[phần chương trình con]

**END PROGRAM** <tên chương trình>

142

### 3.2 Các ngôn ngữ lập trình song song - Fortran 90



143

Ví dụ: tính số  $\pi$  từ công thức  $\int_0^1 \frac{4}{1+x^2} dx$  bằng NNLT Fortran 90

```

1. INTEGER, PARAMETER :: N = 131000
2. INTEGER, PARAMETER :: LONG = SELECTED_REAL_KIND(13,99)
3. REAL(KIND=LONG) PI, WIDTH
4. INTEGER, DIMENSION(N)::id
5. REAL(KIND=LONG), DIMENSION(N)::X, Y
6. WIDTH = 1.0_LONG/N
7. ID = (/ (I, I = 1, N) /)
8. X = (ID - 0.5) * WIDTH
9. Y = 4.0 / (1.0 + X * X)
10. PI = SUM(Y) * WIDTH
11.10 FORMAT('PI = ', F14.12)
12.PRINT 10, PI
13.END

```

Annotations for the code:

- Line 1: khai báo tham số N là số các đoạn con tối đa
- Line 2: khai báo tham số LONG để sử dụng với N phần tử kiểu số nguyên
- Line 3: Khai báo biến mảng id có N phần tử kiểu số nguyên
- Line 4: khai báo biến mảng Y gồm N phần tử khai báo ở dòng 5, thực với mỗi thuộc biến dòng
- Line 5: tính toán song song điểm giữa của mỗi khoảng con
- Line 6: tính toán song song giá trị hàm tương ứng tại các điểm giữa X
- Line 10: gọi hàm tính tổng SUM

144



Tiểu luận: Fortran 90 Tutorial (02 học viên)

Professor Dr.Shene

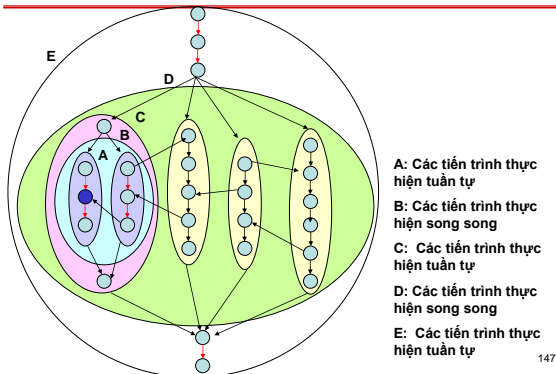
Department of Computer Science  
Michigan Technological University

[www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortran.html](http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortran.html)

<http://www.fortran-2000.com/>

145

### Mô phỏng mô hình lập trình trong Occam



147

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Các cấu trúc điều khiển

**SEQ:** cấu trúc tuần tự, các thành phần trong tiến trình này được thực hiện theo thứ tự và tiến trình này kết thúc khi thành phần cuối cùng thực hiện xong. Mỗi thành phần của tiến trình này có thể là một tiến trình song song.

Ví dụ:

**SEQ**

```
channel1 ? partial1
channel2 ? partial2
sum:= partial1 + partial2
partial3 ! Sum
```

Các số nguyên được nhập vào tuần tự từ 2 kênh khác nhau sau đó cộng lại và chuyển sang kênh 3

149

### 3.2 Các ngôn ngữ lập trình song song - OCCAM

#### 3.2.3 Lập trình song song với OCCAM

##### Giới thiệu về OCCAM:

- NNLT song song được Inmos Company (Anh) phát triển năm 1988,
- Mục đích chính là để thiết kế và cài đặt các chip được gọi là transputer.
- Chương trình Occam thường nhiều tiến trình và chúng có thể được ánh xạ sang một số các transputer bất kỳ để thực hiện song song và trao đổi dữ liệu với nhau thông qua các kênh vào/ra.
- Occam là NNLT bậc cao, được sử dụng để lập trình cho những hệ thống gồm nhiều máy tính kết nối với nhau, hoặc các hệ phân tán.
- Trong Occam, các hành động có thể thực hiện song song được gọi là tiến trình và mỗi câu lệnh cần phải khai báo như một tiến trình.

146

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Các tiến trình trong Occam

Có ba tiến trình nguyên thủy trong Occam:

- Tiến trình gán:** thay đổi giá trị của các biến
- Tiến trình Input:** nhận dữ liệu vào từ các kênh vào (cổng vào)
- Tiến trình Output:** gửi dữ liệu ra các kênh ra.

#### Cấu trúc ngôn ngữ

- Tiến trình gán:**  

```
sum := partion1 + partion2
/*gán gtrj cho biến sum là tổng hai gtrj partion1 và partion2 */
```
- Tiến trình Input:**  

```
user1 ? x
/*gán giá trị từ kênh user1 cho x */
```
- Tiến trình Output:**  

```
C ! x * x
/*gửi giá trị x^2 ra kênh C */
```

148

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Các cấu trúc điều khiển (cont.)

**PAR:** cấu trúc song song, các thành phần của tiến trình này được thực hiện đồng thời và tiến trình này sẽ kết thúc khi tất cả các thành phần của nó đều kết thúc.

Ví dụ:

**SEQ**

**PAR**

```
channel1 ? partial1
channel2 ? partial2
sum:= partial1 + partial2
```

Các số nguyên được nhập vào song song từ 2 kênh khác nhau sau đó cộng lại

150

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Các cấu trúc điều khiển (cont.)

**ALT:** cấu trúc tuyển chọn, chọn một trong các thành phần của tiến trình để thực hiện nếu nó thỏa mãn điều kiện lựa chọn và tiến trình này kết thúc khi thành phần được lựa chọn kết thúc. (xem ví dụ tính tích phân)

151

Ví dụ: Giả sử có hai tiến trình giống nhau cùng nhận dữ liệu vào và cộng dồn vào tổng.

```
CHAN In1, In2:
CHAN Out1, Out2:
VAR Sum1, Sum2:
SEQ
    &các lệnh dưới đây thực hiện tuần tự

    Sum1 := 0      &gán giá trị đầu cho Sum1 và Sum2
    Sum2 := 0
    PAR
        -- Tiến trình thứ nhất
        While TRUE
            VAR Item1:
            SEQ
                In1 ? Item1      &gán giá trị ở kênh In1 cho Item1
                Sum1 := Sum1 + Item1
                Out1 ! Item1
        -- Tiến trình thứ hai
        While TRUE
            VAR Item1:
            SEQ
                In2 ? Item2
                Sum2 := Sum2 + Item2
                Out2 ! Item2
```

153

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Các cấu trúc khác

- Cấu trúc điều khiển **IF**, gọi là *tiến trình điều kiện* để chọn một tiến trình thành phần khi biểu thức Boolean có giá trị *true*;
- Cấu trúc lặp **WHILE**, gọi là *tiến trình lặp* để thực hiện lặp lại tiến trình thành phần cho đến khi biểu thức điều kiện Boolean nhận giá trị *true*.
- Cấu trúc lặp **FOR**, gọi là *tiến trình lặp có số vòng lặp xác định trước*,  $i = [0 \text{ FOR } n]$   
Ví dụ: PAR  $i = [0 \text{ FOR } 9]$ : tạo ra 10 tiến trình song song,  $i$  nhận giá trị từ 0 đến 9

152

### 3.2 Các ngôn ngữ lập trình song song - Occam

#### Sự trao đổi giữa các tiến trình

- Trong ví dụ trên, các tiến trình không cần trao đổi với nhau vì mỗi tiến trình đều sử dụng các biến cục bộ của riêng nó.
- Tuy nhiên, khi có nhiều tiến trình muốn trao đổi dữ liệu với nhau thì phải trao đổi trên cùng một kênh truyền dữ liệu. Nghĩa là, một tiến trình gửi dữ liệu ra một kênh truyền và tiến trình kia nhận dữ liệu từ kênh truyền đó.
- Trong Occam, mỗi tiến trình thực hiện trên một bộ xử lý và truyền thông điệp (Passed-Message) tới những bộ xử lý lân cận theo kiến trúc hình khối.

154

Ví dụ: tính số  $\pi$  từ công thức  $\int_0^1 \frac{4}{1+x^2} dx$  bằng NNLT Occam

- DEF N = 400000:
- DEF PROCESS = 8:
- DEF CHUNK = N / PROCESS:
- CHAN sum[PROCESS]
- PAR
  - PAR  $i = [0 \text{ FOR } \text{PROCESS}]$
  - REAL64x, localsum, width:
  - SEQ
  - localsum := 0.0
  - width := 1.0 / N
  - $x := ((i * \text{CHUNK}) + 0.5) * \text{width}$
  - SEQ  $i = [0 \text{ FOR } \text{CHUNK}]$
  - SEQ
  - localsum := localsum + (4.0 / (1.0 + (x\*x)))
  - $x := x + \text{width}$

Từ dòng 1-3:  
định nghĩa các hằng:  
N là số khoảng chia.  
Dòng 4: định nghĩa một n  
Từ dòng 5-31: là những tiến trình song song và tuần tự, gồm có (PROCESS+1) tiến trình. Mỗi tiến trình trong các tiến trình thực hiện song song từ dòng 5-17, sẽ xác định tổng diện tích các hình chữ nhật được chia sẻ

cont. 155

Ví dụ: tính số  $\pi$  từ công thức  $\int_0^1 \frac{4}{1+x^2} dx$  bằng NNLT Occam

- localsum := localsum \* width -
- sum[i] ! localsum
- REAL64 pi :
- INT got [PROCESS] :
- SEQ
- pi := 0.0
- SEQ  $i = [0 \text{ FOR } \text{PROCESS}]$
- got[i] := FALSE
- SEQ  $i = [0 \text{ FOR } \text{PROCESS}]$
- REAL64 y:
- SEQ
- ALT  $i = [0 \text{ FOR } \text{PROCESS}]$
- (got[i] = FALSE) & sum[i] ? Y
- got[i] := TRUE
- pi := pi + y
- Output ! "Approximation to pi is: "; pi

Từ dòng 18-31: là tiến trình cuối cùng, để xác định tổng sau cùng và chuyển cho tiến trình output để in kết quả pi

156

Tiểu luận (02 học viên):

OCCAM Language

[http://en.wikipedia.org/wiki/Occam\\_programming\\_language](http://en.wikipedia.org/wiki/Occam_programming_language)

<http://www.wotug.org/>

Tiểu luận (02 học viên): ThreadMentor

<http://cs.mtu.edu/~shene/NSF-3/e-Book/index.html>

157

### 3.2 Các ngôn ngữ lập trình song song - PVM

Các đặc điểm chính của PVM:

- Thực hiện theo mô hình truyền thông điệp (*Message Passing*)
- Hỗ trợ sự kết nối không thuần nhất (Heterogeneity)*: PVM hỗ trợ sự kết nối của nhiều máy tính, nhiều mạng máy tính và nhiều loại chương trình ứng dụng khác nhau.
- Hỗ trợ đa bộ xử lý*: PVM sử dụng những khả năng truyền thông điệp trong hệ đa bộ xử lý để khai thác hết khả năng của phần cứng.
- Tính toán dựa trên tiến trình*: Đơn vị điều khiển thực hiện song song trong PVM là một tác vụ, đó là một luồng (*thread*) làm nhiệm vụ điều khiển sự truyền thông và tính toán.
- Thay đổi cấu hình theo yêu cầu*: Các chương trình có thể thực hiện trên tập các máy được lựa chọn theo yêu cầu của NSD.

159

### 3.2 Các ngôn ngữ lập trình song song - PVM

#### 3.2.4 Lập trình song song với PVM (Parallel Virtual Machine)

- PVM được phát triển bởi University of Tennessee, Oak Ridge National Laboratory and Emory University (1989)
- PVM là một phần mềm sử dụng cho hệ thống bao gồm các máy tính không thuần nhất (heterogeneous) được kết nối với nhau để xử lý song song.
- PVM thường được sử dụng cho những máy tính nối mạng trong môi trường UNIX hoặc Windows

158

### 3.2 Các ngôn ngữ lập trình song song - PVM

PVM xử lý tất cả các vấn đề:

- định tuyến truyền thông điệp
- chuyển đổi dữ liệu
- lập lịch trong mạng máy tính

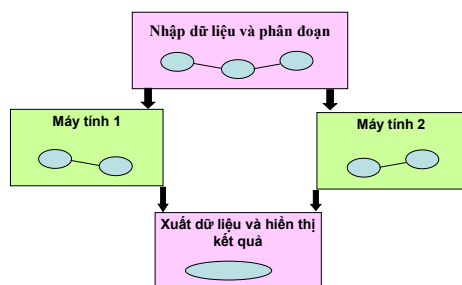
Hệ thống PVM gồm hai thành phần chính:

1. Khối **pvm**d (hoặc **pvm3**) đặt thường trú trên tất cả các máy tính để tạo ra máy ảo (các BXL giả lập).
2. *Thư viện các chương trình con giao diện của pvm*: chứa các chương trình con để truyền thông điệp, quản lý các tiến trình, phối hợp các tác vụ và thay đổi các máy ảo.

160

### 3.2 Các ngôn ngữ lập trình song song - PVM

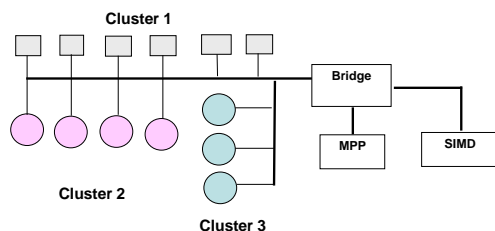
Mô hình tính toán của PVM



161

### 3.2 Các ngôn ngữ lập trình song song - PVM

Một kiến trúc của PVM



162

### 3.2 Các ngôn ngữ lập trình song song - PVM

#### Phương thức thực hiện chương trình trong PVM:

- Những chương trình viết bằng C/C++, Fortran 77 có thể chứa những lời gọi các hàm thư viện của PVM (đây là những ngôn ngữ lập trình được PVM hỗ trợ)
- Các chương trình được dịch theo kiến trúc của hệ thống (*host pool*), các tệp mã đích (object file) được đặt vào những nơi mà mọi máy tính của hệ thống đều truy cập được.
- Các USER tạo ra bản sao của tác vụ chủ (master) hoặc khởi động một tác vụ để thực hiện các ứng dụng của mình.
- Một tiến trình được khởi động bởi một tiến trình khác được gọi là tiến trình tớ (slave).
- Những tiến trình này có thể thực hiện một số tính toán cục bộ và trao đổi với nhau để giải quyết bài toán đặt ra.

163

### 3.2 Các ngôn ngữ lập trình song song - PVM

#### Giới thiệu một chương trình trên PVM

Chương trình in ra màn hình (master) định danh của tác vụ (*ID task*) nhận được từ hàm **pvm\_mytid()**,  
Sau đó sử dụng các hàm:

- pvm\_spawn()**: tạo ra một bản sao và gọi chương trình **hello\_other**
- pvm\_recv()**: nhận và thực hiện khối thông điệp gửi đến.
- pvm\_exit()**: kết thúc chương trình trong hệ thống PVM.

165

### 3.2 Các ngôn ngữ lập trình song song - PVM

#### Một vài thông tin về PVM:

- User muốn chạy một ứng dụng trên PVM thì phải khởi động PVM và tạo ra một máy ảo.
- Một ứng dụng PVM có thể được gọi từ đầu nhắc của hệ điều hành UNIX ở một host nào đó
- Các Users có thể cấu hình máy ảo và thực hiện các ứng dụng của mình với PVM
- Trong PVM mọi tiến trình đều có thể giao tiếp và/hoặc đồng bộ với những tiến trình khác
- Các files PVM có thể truy cập ở địa chỉ:  
<http://www.netlib.org/pvm3/index.html>
- Nhiều máy tính hỗ trợ hệ thống PVM thực hiện như: BBN Butterfly TC2000, 80386/486 PC chạy với UNIX, Thinking Machine's CM-2, CM-5, Cray-2, Cray-5MP, HP-9000 PA-RISC, Intel Paragon, Silicon Graphics IRIS, Sun 4, DEC Micro VAX, v.v.

164

### 3.2 Các ngôn ngữ lập trình song song - PVM

#### Giới thiệu một chương trình trên PVM (cont.)

Chương trình ở slave:

**pvm\_parent()** để xác định *ID task* của master.

Sau đó dùng các hàm dưới đây để xác định hostname và chuyển hostname này cho master:

**pvm\_initsend()**: khởi tạo buffer để gửi

**pvm\_pkstr()**: đặt một xâu vào buffer để gửi đi

**pvm\_upkstr()**: đọc một xâu vào buffer

**pvm\_send()**: chuyển dữ liệu ở buffer tới tiến trình nhận được xác định bởi **pvm\_mptid**.

166

### 3.2 Các ngôn ngữ lập trình song song - PVM

```

/* Chương trình master có tên Hello.c */
#include "pvm3.h"
main() {
    int cc, tid, msg;
    char buf[100];
    printf("Master ID number %x\n", pvm_mytid());
    cc = pvm_spawn("Hello_other", (char**)0, 0, "", 1, &tid);
    if(cc == 1) {
        msg = 1;
        pvm_recv(tid, msg);
        pvm_upkstr(buf);
        printf("From master %x: %s\n", tid, buf);
    } else printf("Cannot start hello_other\n");
    pvm_exit();
}

```

tạo ra một bản sao và gọi chương trình **hello\_other**

nhận và thực hiện khối thông điệp gửi đến

đọc một xâu vào buffer

kết thúc chương trình trong hệ thống PVM

### 3.2 Các ngôn ngữ lập trình song song - PVM

```

/* Chương trình slave có tên: hello_other.c */
#include "pvm3.h"
main() {
    int ptid, msg;
    char buf[100];
    ptid = pvm_parent();
    strcpy(buf, "Hello world from ");
    gethostname(buf + strlen(buf), 64);
    msg = 1;
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, msg);
    pvm_exit();
}

```

**pvm\_parent()** nhận một tác vụ của tiến trình tớ từ tiến trình chủ

**pvm\_initsend()** khởi tạo buffer để gửi

**pvm\_pkstr()** đặt một xâu vào buffer để gửi đi

**pvm\_send()** chuyển dữ liệu ở buffer tới tiến trình nhận được xác định bởi **ptid**.

168

## Tiểu luận

## 1. Tìm hiểu về PVM

<http://www.csm.ornl.gov/pvm/intro.html>  
<http://www.csm.ornl.gov/pvm/>  
<http://search.cpan.org/dist/Parallel-Pvm/>  
<http://www.parallels.com/>  
<http://www.netlib.org/pvm3/book/node1.html>

169

## Tiểu luận

## 2. Thread trong JAVA

– <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Thread.html>  
 – <http://en.wikipedia.org/wiki/Multithreading>

Một số ví dụ về cách tạo thread trong JAVA

• <http://www.niitvn.com/niitwcmis/dt/QLT/MotcachtaoThreadtrongJava.pdf>

• <http://www.niit-n.com/4rum/showthread.php?p=8754>

Introduction to Java threads:

<http://www.javaworld.com/javaworld/jw-04-1996/jw-04-threads.html>

170

## 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

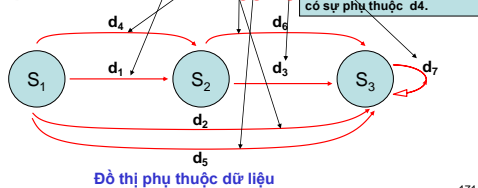
## Mục đích:

- Xác định mức độ phụ thuộc dữ liệu giữa các tiến trình
- Ví dụ: Xét dãy các lệnh sau:

$S_1: A := B + C$

$S_2: B := A + E$

$S_3: A := A + B$



171

## 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

## Đồ thị phụ thuộc dữ liệu?

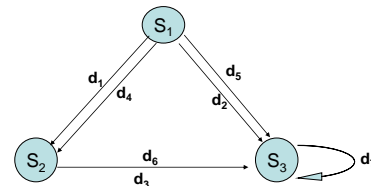
- là một đồ thị có hướng  $G=(V,E)$ , trong đó  $V$  là tập các lệnh trong chương trình,  $E$  là các phụ thuộc dữ liệu.

## • Ví dụ

$S_1: A := B + C$

$S_2: B := A + E$

$S_3: A := A + B$



172

## 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

## Ký hiệu:

**DEF(S)** - tập tất cả các biến có giá trị bị thay đổi khi thực hiện câu lệnh S.

**USE(S)** - tập tất cả các biến được truy cập (được sử dụng) khi thực hiện câu lệnh S.

Ví dụ:  $S_1: A := B + C$      $S_2: B := A + E$      $S_3: A := A + B$

$DEF(S_1) = \{A\}$

$DEF(S_2) = \{B\}$

$DEF(S_3) = \{A\}$

$USE(S_1) = \{B, C\}$

$USE(S_2) = \{A, E\}$

$USE(S_3) = \{A, B\}$

173

## 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

## Phân loại các phụ thuộc dữ liệu:

Xét các câu lệnh  $S_1: A := B + C$ ;  $S_2: B := A + E$ ;  $S_3: A := A + B$

## 1. Phụ thuộc dòng dữ liệu (Data Flow Dependency):

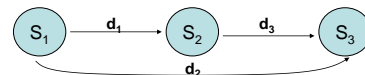
- là sự phụ thuộc dữ liệu giữa  $S_1$  và  $S_2$  khi  $DEF(S_1) \cap USE(S_2) \neq \emptyset$

• Đây là loại phụ thuộc rất chung và rất khó loại bỏ bởi vì lệnh  $S_2$  yêu cầu giá trị của một biến và giá trị này phải được tính ở  $S_1$ .

• Khi xuất hiện phụ thuộc dòng dữ liệu giữa các câu lệnh thì chúng **không thực hiện song song được**.

• Quan hệ phụ thuộc dòng dữ liệu được ký hiệu

• Ví dụ: các phụ thuộc  $d_1, d_2, d_3$  là loại phụ thuộc dòng dữ liệu.



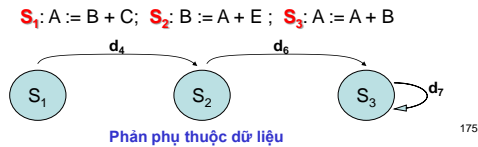
Phụ thuộc dòng dữ liệu

174

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### 2. Phân phụ thuộc dữ liệu (Data Anti-Dependency):

- là sự phụ thuộc dữ liệu giữa  $S_1$  và  $S_2$  khi  $DEF(S_2) \cap USE(S_1) \neq \emptyset$
- Sự phụ thuộc này xuất hiện khi chúng ta sử dụng lại tên gọi của các biến, một biến đã được sử dụng trong  $S_1$  và sau đó được định nghĩa lại ở  $S_2$ .
- Khi xuất hiện phân phụ thuộc dữ liệu giữa các câu lệnh thì chúng cũng **không thực hiện song song được**.
- Quan hệ phân phụ thuộc dữ liệu được ký hiệu:
- Ví dụ: các phụ thuộc  $d_4, d_6, d_7$  là các **phân phụ thuộc dữ liệu**.



175

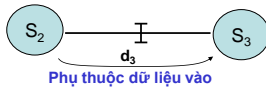
### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### 4. Phụ thuộc dữ liệu vào (Data Input Dependency):

- là sự phụ thuộc dữ liệu giữa  $S_1$  và  $S_2$  khi  $USE(S_2) \cap USE(S_1) \neq \emptyset$
- Bởi vì các lệnh này chỉ truy cập (đọc) và không làm thay đổi giá trị của các biến đó, do vậy các lệnh này có thể thực hiện theo bất kỳ thứ tự nào cũng được, nghĩa là **có thể thực hiện song song**.
- Quan hệ phụ thuộc dữ liệu vào được ký hiệu:
- Ví dụ

$S_2: B := A + E; S_3: A := A + B$

$USE(S_3) \cap USE(S_2) = \{A\} \neq \emptyset$

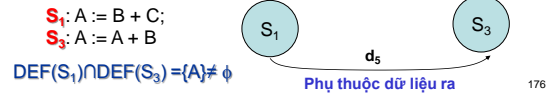


177

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### 3. Phụ thuộc dữ liệu ra (Data Output Dependency):

- là sự phụ thuộc dữ liệu giữa  $S_1$  và  $S_2$  khi  $DEF(S_2) \cap DEF(S_1) \neq \emptyset$
- Sự phụ thuộc này xuất hiện do hai nguyên nhân:
  - Sử dụng lại tên của các biến (dùng chung)
  - Tính tăng giá trị của cùng một biến.
- Nếu những lệnh này thực hiện đồng thời thì chúng sẽ ghi đè các giá trị vào cùng một ô nhớ. Do vậy, cần phải xác định chính xác thứ tự thực hiện để ngăn ngừa việc sử dụng những giá trị không đúng.
- Quan hệ phụ thuộc dữ liệu ra được ký hiệu là:
- Ví dụ: phụ thuộc  $d_5$  là loại phụ thuộc dữ liệu ra.



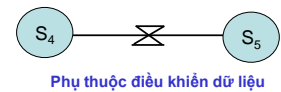
176

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### 5. Phụ thuộc điều khiển dữ liệu (Data Control Dependency):

- là sự phụ thuộc dữ liệu giữa  $S_1$  và  $S_2$  khi sự thực hiện của lệnh này phụ thuộc vào giá trị của các biến được tính ở lệnh kia.
- Quan hệ phụ thuộc điều khiển dữ liệu được ký hiệu là:

$S_4: P = (B \Rightarrow 0)$   
 $S_5: \text{if } (P \text{ is true}) \text{ then}$   
            $D = 1$   
       else  $D = 2$   
       endif



178

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### Nhận xét:

- Để tăng được mức độ song song của chương trình chúng ta có thể **khử** hoặc **giảm** các **phân phụ thuộc dữ liệu** và **phụ thuộc dữ liệu ra** trong đồ thị phụ thuộc dữ liệu.
- Thông thường để **khử** các phụ thuộc dữ liệu này là **đặt lại tên cho các biến để tránh việc chia sẻ các biến đó**.
- Ví dụ: Xét đoạn chương trình sau:

<pre>for i = 1, n, 1   X = A[i] + B[i]   Y[i] = 2 * X   X = C[i] * D[i]   P = X + 15 endfor</pre>		<pre>for i = 1, n, 1   X = A[i] + B[i]   Y[i] = 2 * X   XX = C[i] * D[i]   P = XX + 15 endfor</pre>
---	--	---

#### Nhận xét:

- Có phân phụ thuộc dữ liệu. Nếu hai lệnh sau ta đặt X thành XX thì sẽ tạo ra hai đoạn chương trình có thể thực hiện song song:

179

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

- Sử dụng các biến khác nhau để loại bỏ **phụ thuộc dữ liệu ra**.
- Ví dụ:

<pre>A = B + C A = C * X</pre>		<pre>A = B + C A1 = C * X</pre>
--------------------------------	--	---------------------------------

$DEF(S_2) \cap DEF(S_1) = \{A\} \neq \emptyset$

#### Nhận xét:

- Hai lệnh trên có sự phụ thuộc dữ liệu ra.
- Có thể loại bỏ phụ thuộc này bằng cách thay biến A của câu lệnh thứ hai bằng  $A_1$

180

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

+ Sử dụng các biến khác nhau để loại bỏ *phản phụ thuộc dữ liệu*

Ví dụ:

$$\begin{array}{l} A = B + C \\ B = C * X \end{array} \Rightarrow \begin{array}{l} A = B + C \\ B1 = C * X \end{array}$$

$$DEF(S_2) \cap USE(S_1) = \{B\} \neq \emptyset$$

Nhận xét:

- Hai lệnh trên có quan hệ phản phụ thuộc dữ liệu.
- Có thể loại bỏ sự phụ thuộc này bằng cách sử dụng một biến khác cho câu lệnh thứ hai

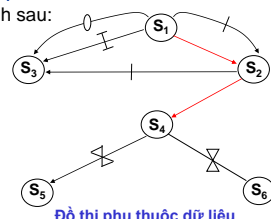
181

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

+ Thực hiện một số phép biến đổi (phép thế) cũng có thể loại bỏ được *phản phụ thuộc dữ liệu*.

Ví dụ: Xét dãy các câu lệnh sau:

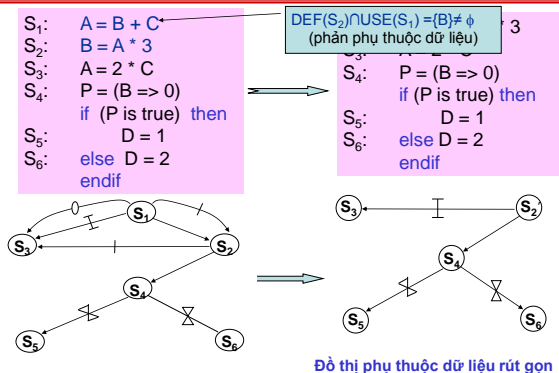
```
S1: A = B + C
S2: B = A * 3
S3: A = 2 * C
S4: P = (B => 0)
      if (P is true) then
S5:       D = 1
S6:       else D = 2
      endif
```



Đồ thị phụ thuộc dữ liệu

Nhận xét: Để xử lý song song, thì cần thiết phải loại bỏ đi một số loại phụ thuộc dữ liệu có thể.

Ví dụ: loại bỏ những quan hệ phản phụ thuộc dữ liệu và phụ thuộc dữ liệu kết quả



Đồ thị phụ thuộc dữ liệu rút gọn

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

Tuy nhiên, những phép biến đổi đơn giản như thế không phải lúc nào cũng áp dụng thành công. Ví dụ, xét chu trình sau:

```
for (i = 0; i < N; i++)
  A = A + i;
```

không sử dụng FOR

```
i = 0;
aa: if(i > N) stop;
   a = a + i;
   i++;
   goto aa;
```

Nhận xét:

- Trong mỗi bước lặp đều phải sử dụng những tên biến khác nhau nếu muốn loại bỏ phản phụ thuộc dữ liệu. Điều này khó thực hiện được trong trường hợp số vòng lặp lớn hoặc bất kỳ.
- Quan hệ phụ thuộc dữ liệu là không có tính bắc cầu. Nghĩa là nếu  $S_2$  phụ thuộc vào  $S_1$ ,  $S_3$  phụ thuộc vào  $S_2$  thì không kết luận được  $S_3$  phụ thuộc dữ liệu vào  $S_1$ .

184

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

**Phụ thuộc theo chu trình và theo mảng**

**Mục đích:** khử phụ thuộc dữ liệu trong các vòng lặp.

Xét đoạn chương trình sau:

```
DO i
  A[2*i+1] = B[i]      (1)
  D[i] = A[2*i]        (2)
ENDDO
```



hai câu lệnh (1) và (2) có thực hiện song song được không?

• Nếu xem mảng A như một thực thể đơn thì hai lệnh trên là phụ thuộc theo dòng dữ liệu (vì A xuất hiện trong DEF của (1) và cũng xuất hiện trong USE của (2)).

• Nhưng nếu ta xét thêm chỉ số mảng thì trong câu lệnh (1) sử dụng các phần tử mảng A có chỉ số lẻ; còn ở (2) lại sử dụng phần tử mảng A có chỉ số chẵn.

Do đó, ở đây không có sự phụ thuộc dữ liệu.

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

**Vấn đề:** khi xây dựng các chu trình lặp phải xét xem những dãy lặp khác nhau của chúng có thể thực hiện song song được không?

Nói cách khác, khi một bộ xử lý P thực hiện một dãy các câu lệnh với  $i = 1$  (biến điều khiển vòng lặp) thì bộ xử lý Q có thể cùng thực hiện dãy các câu lệnh đó với  $i=2$  hay không?

Xét ví dụ sau:

```
FOR I=1 DO
  A[I] = A[I-1] + 1      (3)
```

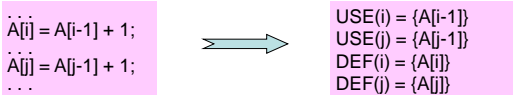
Chu trình trên có thể khai triển thành:

```
A[1] = A[0] + 1; A[2] = A[1] + 1;
A[3] = A[2] + 1; A[4] = A[3] + 1; . . .
```

186

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

Xét hai vòng lặp thứ  $i$  và thứ  $j$  tương ứng với  $l = i$  và  $l = j$ . Không mất tính tổng quát ta có thể giả thiết  $i < j$ . Chu trình sẽ có dạng:



#### Nhận xét:

- Các tập trên không chứa biến đơn mà chỉ có các biến có chỉ số và khi  $i$  và  $j$  khác nhau thì các biến đó khác nhau.
- Hai tập  $DEF(i) = \{A[i]\}$  và  $USE(j) = \{A[j-1]\}$  có chứa phần tử chung khi  $i = j - 1$ . Vì phương trình  $i = j - 1$  có vô số nghiệm

• Vậy kết luận chu trình trên không thực hiện song song được.

187

### 3.3 Sự phụ thuộc dữ liệu và đồ thị ưu tiên của các tiến trình

#### Nhận xét:

- Nói chung, việc xác định sự phụ thuộc theo cách như trên là bài toán khó. Sự phức tạp sẽ xuất hiện bởi vì có nhiều dạng phương trình không hiển nhiên. Những phương trình này không có lời giải trực tiếp trong các công cụ toán học kinh điển.
- Trường hợp tổng quát, khi xét  $n$  vòng lặp mà dẫn đến dạng phương trình  $a_1x_1 + a_2x_2 + \dots + a_nx_n = c$  với  $a_i$  và  $c$  là các số nguyên (gọi là phương trình Diophantine) thì kết luận chu trình không thể thực hiện song song.

Lưu ý: Phương trình Diophantine có dạng tổng quát:

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = c \text{ với } a_i \text{ và } c \text{ là các số nguyên.}$$

Năm 1991 Zima và Chapman đã chứng minh được rằng nếu ước số chung lớn nhất  $(a_1, a_2, \dots, a_n)$  chia hết cho  $c$  thì phương trình Diophantine có một nghiệm nguyên.

Trường hợp tổng quát, giải phương trình trên là không thể.

188

### 3.4 Biến đổi chương trình

#### Mục đích:

- Loại bỏ được sự phụ thuộc của các câu lệnh trong ch. trình.
- Khắc phục được những trở ngại của cách xác định các vòng lặp có thể thực hiện song song dẫn tới phương trình Diophantine.

#### 3.4.1 Các biến qui nạp

Biến qui nạp là loại biến trong chu trình mà các giá trị liên tiếp của nó tạo ra dãy cấp số cộng.

Ví dụ: xét đoạn chương trình sau:

```

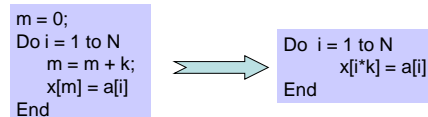
m = 0;
DO i = 1 to N
    m = m + k;
    x[m] = a[i]
END

```

{m là biến qui nạp}

189

### 3.4 Biến đổi chương trình



#### Nhận xét:

- Hai câu lệnh trên có sự phụ thuộc dữ liệu vì  $m$  được sử dụng chéo giữa các vòng lặp.
- Khi vòng lặp trước tính giá trị của  $m$  thì  $m$  lại được sử dụng ở lệnh thứ hai nên chúng phải thực hiện tuần tự.
- Nếu mỗi vòng lặp chúng ta dự báo được giá trị của  $m$  thì có thể sẽ loại bỏ được sự phụ thuộc dữ liệu liên quan đến  $m$ .
- Giá trị  $k$  là bất biến trong các vòng lặp và được cộng liên tiếp vào sau mỗi vòng lặp, do vậy, ở vòng lặp thứ  $i$ ,  $m = i * k$ .

190

### 3.4 Biến đổi chương trình

#### 3.4.2 Sự phụ thuộc tiến

Phụ thuộc tiến là loại **phần phụ thuộc dữ liệu** giữa các vòng lặp của chu trình.



#### Nhận xét:

- Chu trình không thực hiện song song được
- Chu trình làm nhiệm vụ sao tất cả các phần tử của một mảng sang chính mảng đó với chỉ số giảm đi một, do vậy nó phải thực hiện tuần tự.

#### Cách giải quyết

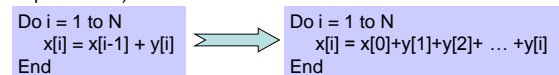
Sử dụng 2 mảng: một bản gốc và một bản sao của mảng gốc để viết lại chương trình

191

### 3.4 Biến đổi chương trình

#### 3.4.3 Sự phụ thuộc lùi

Trường hợp các phần tử của mảng trong chu trình lại được xác định thông qua những phần tử đứng trước chúng (lùi về phía trước)



#### Nhận xét:

- Việc loại bỏ những phụ thuộc dữ liệu kiểu này không đơn giản
- Không có giải pháp chung để loại bỏ phụ thuộc dữ liệu trong trường hợp tổng quát. Nếu có giải pháp loại bỏ thì giá phải trả cũng khá đắt.

Hiển nhiên không còn sự phụ thuộc lùi trong chu trình, tuy nhiên thời gian tính toán trong mỗi câu lệnh lại tăng lên khá nhiều.

192



### 3.4 Biến đổi chương trình

#### 3.4.4 Sự phân tách chu trình (phụ thuộc lùi)

Tách một chu trình thành nhiều chu trình con để thực hiện song song

```
Do i = 1 to N
  a[i] = b[i] + c[i];
  c[i] = a[i-1]
End
```



```
Do i = 1 to N
  a[i] = b[i] + c[i];
End
Do i = 1 to N
  c[i] = a[i-1]
End
```

Nhận xét:

- Sự tham chiếu giữa các mảng  $a$  và  $c$  dẫn đến sự phụ thuộc giữa các vòng lặp của chu trình.
- Giá trị  $c[i]$  ở lệnh thứ nhất luôn là những giá trị lưu trữ cũ của mảng  $c$  (được xác định ở lệnh thứ 2)
- $a[i-1]$  được sử dụng ở lệnh thứ hai là giá trị đã được tính ở vòng lặp trước đó (vòng thứ  $i-1$ )

193

### 3.4 Biến đổi chương trình

#### 3.4.4 Sự phân tách chu trình (phụ thuộc tiến)

```
Do i = 1 to N
  a[i] = b[i] + c[i];
  c[i] = a[i+1]
End
```



```
Do i = 1 to N
  x[i] = c[i];
  c[i] = a[i+1]
End
Do i = 1 to N
  a[i] = b[i] + x[i];
End
```

Nhận xét:

- Phần tử  $c[i]$  ở lệnh hai được tính theo giá trị của  $a[i+1]$ , là giá trị trước khi nó được cập nhật ở lệnh thứ nhất
- Ngược lại  $c[i]$  được sử dụng ở lệnh thứ nhất lại là giá trị cũ trước khi được cập nhật ở lệnh thứ hai.

Cách giải quyết

- Sao mảng  $c$  sang mảng phụ  $x$ . Kết quả biến đổi chu trình này thành hai chu trình trong đó không còn sự phụ thuộc dữ liệu

194

### 3.4 Biến đổi chương trình

#### 3.3.5 Các chu trình lồng nhau

Xét trường hợp nhiều chu trình lồng nhau (chu trình nhiều chỉ số)

```
Do i = 1 to N
  Do j = 1 to N
    x[i,j] = x[i, j-1] + z[i];
  End
```

Nhận xét:

- Mảng  $z$  được sử dụng trong chu trình hoàn toàn không gây ra sự phụ thuộc nào cả.
- Chỉ số thứ nhất của mảng  $x$  là  $i$ , cũng chính là chỉ số của chu trình, nhưng không có sự tham chiếu chéo giữa các vòng lặp. Vậy, chu trình lặp bên trong (chu trình lặp theo  $j$ ) có thể thực hiện song song.
- Chỉ số thứ hai của mảng  $x$  là  $j$ , là chỉ số của chu trình thứ nhất do đó có sự phụ thuộc dòng dữ liệu của các lần lặp ở vòng ngoài. Cũng như ở 3.4.3, trường hợp này không có cách biến đổi đơn giản về dạng khả song song.

195

### Bài tập

3.1 Xác định sự phụ thuộc dữ liệu của các lệnh trong chu trình sau:

```
Do i = 1 to N
  e[i] = x[i] - z[i];
  a[i+1] = e[i] + 2*d[i]
  a[i] = e[i]
end
```

3.2 Hai chu trình sau có tương đương về nội dung tính toán hay không? hãy bình luận về khả năng thực hiện song song của các chu trình đó.

- Do  $i = 1$  to  $N$   
 $a[i] = a[i+1] + i$   
 end
- Do  $i = N$  downto  $1$   
 $a[i] = a[i+1] + i$   
 end

196

### Bài tập

3.3 Xác định tất cả các sự phụ thuộc dữ liệu trong đoạn chương trình sau:

```
int a[MAX];
read(a);
for(i = 0; i < N; i++){
  for(j = 0; j < i; j++){
    a[i] = max(a[i], a[j]);
    a[j] = min(a[i], a[j]);
  }
}
```

3.4 Loại bỏ các phụ thuộc dữ liệu ra và phân phụ thuộc dữ liệu của chu trình sau

```
for(i = 0; i < N; i++){
  x = a[i] + b[i];
  y[i] = 2 * x;
}
```

197

### Bài tập

3.5 Phân tích đoạn chương trình sau, xác định các phụ thuộc dữ liệu và vẽ đồ thị phụ thuộc dữ liệu của đoạn chương trình đó.

```
A = B + C;
for(i = 0; i < N; i++){
  D[i] = A + b[i];
  S = b[i] * 5;
  T = T + S;
}
```

3.6 Viết chương trình giải phương trình bậc hai và vẽ đồ thị phụ thuộc dữ liệu của nó.

198

## CHƯƠNG 4. CÁC MÔ HÌNH VỀ LẬP TRÌNH SONG SONG

## NỘI DUNG

1. Lập trình bộ nhớ chia sẻ
2. Lập trình song song dựa vào các tiến trình
3. Lập trình song song dựa vào các luồng
4. Lập trình theo mô hình truyền thông điệp.
5. Lập trình trên cụm máy tính với PVM

199

## 1. LẬP TRÌNH BỘ NHỚ CHIA SẺ

## Một vài chú ý (2/4)

5. Về nguyên tắc, chương trình là tập các tiến trình và việc viết chương trình là độc lập với các bộ xử lý, việc phân chia các tiến trình cho các bộ xử lý là công việc của hệ điều hành..
6. Trong những hệ thống đa nhiệm như UNIX, số bộ xử lý và số các tiến trình không nhất thiết phải bằng nhau. Nhưng nên tổ chức một tiến trình có một BXL đảm nhiệm để tiện việc lập lịch.
7. Một vấn đề quan trọng cần xem xét khi XLSS là cần bao nhiêu nút (BXL) để chương trình song song thực hiện hiệu quả nhất?
8. Nhiều chương trình được xây dựng phụ thuộc vào một cấu hình xác định. Nói chung, loại chương trình phụ thuộc như vậy là không tốt, vì khi bộ xử lý bận thì các tiến trình tiếp theo phải chờ.

201

## 1. LẬP TRÌNH BỘ NHỚ CHIA SẺ

## Một vài chú ý (3/4)

## Trong lập trình bộ nhớ chia sẻ:

- Các tác vụ (tasks) sẽ đọc/ghi dữ liệu từ bộ nhớ chia sẻ qua không gian địa chỉ bộ nhớ chung (common address space)
- Có những cơ chế khác nhau như (locks/semaphores) để điều khiển việc truy nhập đến bộ nhớ chia sẻ
- Người lập trình không cần mô tả việc truyền thông dữ liệu, do đó việc viết chương trình sẽ đơn giản.
- Khó quản lý dữ liệu vì không can thiệp trực tiếp quá trình truyền dữ liệu.

203

## 1. LẬP TRÌNH BỘ NHỚ CHIA SẺ

## Một vài chú ý (1/4)

1. Hệ thống đa bộ xử lý đối xứng *SMP* (symmetric multiprocessor Sysstem)
  - Các bộ xử lý là như nhau
  - Không có những BXL đặc biệt để xử lý vào/ra
  - Không có BXL được gán nhiệm vụ đặc biệt nào khác.
2. Để nghiên cứu về XLSS, chúng ta không nhất thiết phải có hệ đa bộ xử lý vật lý.
3. Trong môi trường *UNIX*, 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.
4. Hầu hết các hệ UNIX đều cho phép tạo ra một số các tiến trình bất kỳ và chúng được lập lịch cho những bộ xử lý thích hợp

200

## 1. LẬP TRÌNH BỘ NHỚ CHIA SẺ

## Một vài chú ý (3/4)

Trong môi trường lập trình chia sẻ bộ nhớ có hai ràng buộc quan trọng:

1. Một tiến trình có thể chờ một khoảng thời gian bất kỳ giữa hai lệnh cần thực hiện. Giả sử bộ xử lý *P* thực hiện một chương trình có một 100 lệnh, bộ xử lý *Q* thực hiện chương trình có 10 lệnh và cùng bắt đầu thực hiện đồng thời. Thậm chí, tất cả các lệnh có tốc độ thực hiện như nhau cũng không thể nói rằng *Q* sẽ kết thúc trước *P*.
2. Không thể xem các lệnh thực hiện là nguyên tố ở mức các ngôn ngữ lập trình. Ví dụ, một lệnh đơn giản như:  $a = a + 1$  sẽ là một dãy bốn 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.

202

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

## Tạo lập tiến trình

**Cú pháp:** `id = create_process(N);`

**Mục đích:** tạo ra *N* tiến trình và một tiến trình cha để thực hiện câu lệnh đó.

Như thế, có  $N+1$  tiến trình như nhau được tạo ra và mỗi giá trị của *id* được gán tương ứng cho một tiến trình.

204

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

Ví dụ: tạo  $N$  tiến trình và giao  $n+1$  Nhiệm vụ để thực hiện song song

```
id = create_process(N);
switch(id){
    case 0: ... do NhiệmVu0 ...; break;
    case 1: ... do NhiệmVu1 ...; break;
    case 2: ... do NhiệmVu2 ...; break;
    ...
    case N: ... do NhiệmVuN ...; break;
}
```

205

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

*Vấn đề là các tiến trình được tạo lập và truy cập dữ liệu của nhau như thế nào?*

- Một mặt một tiến trình có thể muốn giữ một phần dữ liệu cục bộ cho riêng mình, không cho những tiến trình khác 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 dấu hay chia sẻ thông tin như thế nào còn tùy thuộc vào mô hình mà chúng ta sử dụng: **tiến trình** (process) hay **luồng** (thread).
- Các tiến trình trong UNIX được sử dụng như các đơn vị tính toán độc lập. Theo mặc định, việc tính toán và cập nhật bộ nhớ của một tiến trình là không nhìn thấy được từ các tiến trình khác.
- Đối với luồng, tất cả các thông tin, theo mặc định, là nhìn thấy được.

207

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

### b) Hủy bỏ tiến trình:

- Cú pháp:** `join_process(N, id);`
- Mục đích:** cho phép tiến trình có giá trị `id` trong  $N$  tiến trình đã khởi tạo được tiếp tục thực hiện những phần việc tuần tự còn lại, còn những tiến trình khác kết thúc.

Nếu ta đặt sau `join_process(N, id)` một câu lệnh thì câu lệnh này sẽ không được thực hiện cho đến khi tất cả các tiến trình đều thực hiện `join_process(N, id)`. Do đó vấn đề xử lý song song không xuất hiện mà là xử lý tuần tự.

206

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

### Chú ý (1/3):

- Khi muốn sử dụng bộ nhớ chung, Người lập trình cần phải xin cấp phát bộ nhớ và sau khi sử dụng xong phải giải phóng chúng.

Có hai hàm cơ sở để thực hiện điều này:

- > `shared(m, &id)`:** xin cấp phát  $m$  byte bộ nhớ chia sẻ cho tiến trình `id`.
- > `free_shm()`:** giải phóng bộ nhớ đã được cấp.

Các hàm `shared(m, &id)`, `free_shm()` được gọi là hàm điều phối về chia sẻ bộ nhớ.

208

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

**Ví dụ:** Bài toán loại trừ nhau.

Xét đoạn chương trình sau:

```
main(){
    int id, sid, *i, j;
    i = (int*)shared(sizeof(int), &sid);
    *i = 100; j = 100;
    printf("Before fork: %d, %d\n", *i, j);
    id = create_process(2);
    *i = id; j = id * 2;
    printf("After fork: &d, %d\n", *i, j);
    join_process(3, id);
    printf("After join: &d, %d\n", *i, j);
    free_shm(sid);
}
```

**Trả lời:**

**Chưa chắc!!!**  
Các giá trị của `*i, j` in ra có thể là 2, 4; 4, 4; 6, 4; bởi vì khi tiến trình thứ  $i$  thực hiện xong câu lệnh (1), trước khi nó thực hiện (2) thì có thể các tiến trình khác đã cập nhật lại giá trị của  $i$ .  
Lưu ý rằng,  $i$  là biến chia sẻ,  $id$  là duy nhất (số hiệu) đối với mỗi tiến trình.

Tạo 2 tiến trình được một tiến trình  
Chỉ tiến giá trị động, kết thúc  
Giải phóng

**Câu hỏi:** Giả sử câu lệnh (1) thực hiện với `id = 2` thì câu lệnh (2) có in ra là "After fork: 2, 4" hay không?

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẺ DỰA VÀO TIẾN TRÌNH

### Chú ý:

- Nếu có một tiến trình truy cập vào một vùng nhớ với ý định cập nhật thì nó phải được đảm bảo rằng không một tiến trình nào khác đọc dữ liệu ở vùng đó cho đến khi việc cập nhật đó kết thúc.
- Để giải quyết được vấn đề trên thì phải có cơ chế đảm bảo rằng, tại mỗi thời điểm các khối lệnh của chương trình được thực thi chỉ bởi một tiến trình.
- Nếu có một tiến trình bắt đầu vào thực hiện một khối lệnh thì những tiến trình khác không được vào khối lệnh đó.
- Khi một tiến trình vào một vùng lệnh nào đó thì nó sẽ **gài khoá** (lock).
- Ngược lại, khi ra khỏi vùng đó thì thực hiện cơ chế **mở khoá** (unlock) để cho tiến trình khác có nhu cầu sử dụng.

210

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO TIẾN TRÌNH

Các câu lệnh để thực hiện các yêu cầu trên:

- **init\_lock(id):** Khởi động bộ khoá vùng nhớ chia sẻ, trong đó *id* là tên của vùng nhớ sử dụng chung.
- **lock(id):** khoá lại vùng nhớ *id*. Nếu một tiến trình đã khoá một vùng nhớ chung thì những tiến trình khác muốn truy cập vào đó sẽ phải chờ.
- **unlock(id):** mở khoá vùng đã bị khoá và trả lại cho tiến trình khác.

211

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO TIẾN TRÌNH

Sử dụng cơ chế *gài khoá* để viết lại chương trình trên:

```
main(){
    int *lock1, id, sid1, sid2, *i, j;
    lock1 = (int*)shared(sizeof(int), &sid1);
    init_lock(lock1);
    i = (int*)shared(sizeof(int), &sid2);
    *i = 100; j = 100;
    printf("Before fork: %d, %d\n", *i, j);
    id = create_process(2);
    lock(lock1);
    *i = id; j = id * 2; // (1)
    printf("After fork: &d, %d\n", *i, j); // (2)
    unlock(lock1);
    join_process(3, id);
    printf("After join: &d, %d\n", *i, j);
    free_shm(sid1); free_shm(sid2);
}
```

212

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO TIẾN TRÌNH

**Ví dụ (1/2):** Chương trình tính tổng của hai vector:

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

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

Giả sử ta có *M* tiến trình. Chúng ta có thể chia *N* phần tử thành *M* phần (giả thiết *N/M* là số nguyên) 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, nhận giá trị từ 0 đến *M-1*. Tiến trình thứ *i* xử lý *N/M* phần tử liên tiếp kể từ *i\*N/M*.

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>5</sub>
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Khi *N = 15* và *M = 5*

213

## 1.1 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO TIẾN TRÌNH

**Ví dụ (2/2)**

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

Tiến trình *P<sub>i</sub>* bắt đầu từ phần tử thứ *i*, sau đó bỏ qua *M* phần tử để xử lý phần tử tiếp theo, nghĩa là nó truy cập đến *i, i+M, i+2M, v.v*

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

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

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>5</sub>
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Khi *N = 15* và *M = 5*

214

## 1.2 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO LUỒNG

### ? tiến trình ≠ luồng

- Một tiến trình là bức tranh về sự hoạt động của một ch.trình.
- Mỗi tiến trình bao gồm 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, 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, nhưng việc đồng bộ các luồng được thực hiện hiệu quả hơn đối với các tiến trình.

• Đồng bộ 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 của chương trình.

215

## 1.2 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO LUỒNG

• Nhiều hệ điều hành hiện nay hỗ trợ đa luồng như:

- SUN Solaris
- Window NT
- OS/2, v.v.

Bên trong những hệ điều hành này, những đặc tính hỗ trợ cho NSD khai thác được các luồng trong chương trình của mình thường khác nhau.

Hiện nay đã có một chuẩn, đó là **Pthread** của IEEE Portable Operating System Interface, POSIX.

216

## 1.2 LẬP TRÌNH BỘ NHỚ CHIA SẼ DỰA VÀO LUỒNG

### Thực hiện các luồng của Pthread

Trong **Pthread**, chương trình chính cũng chính là một luồng. Một luồng có thể được khai báo, tạo lập và kết thúc bằng những chương trình con sau:

```
pthread_t aThread;           // Khai báo một luồng
pthread_create(&aThread,&status,(void*)proc1,(void*)arg);
pthread_join(aThread, void *status);
```

217

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG



Hoạt động của một luồng trong Pthread

218

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

**Ví dụ:** Xét một chương trình đơn giản sử dụng các luồng. Chương trình gồm hai chương trình con *reader()* và *processor()*. *Reader()* đọc dãy dữ liệu từ luồng vào và *processor()* xử lý trên các dữ liệu đọc vào.

xem giáo trình

219

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

Vấn đề thực hiện loại trừ nhau giữa các luồng  
Vấn đề thực hiện loại trừ nhau giữa các luồng cũng tương tự như đối với các tiến trình.  
Có bốn hàm nguyên thủy:

- `pthread_mutex_init(mutex, NULL)`
- `pthread_mutex_lock(mutex)`
- `pthread_mutex_unlock(mutex)`
- `pthread_mutex_destroy(mutex)`

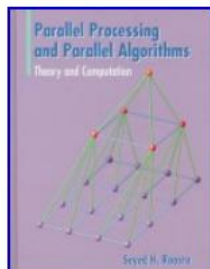
**Ví dụ:** Chương trình đọc vào một mảng *a[]* các số nguyên và thực hiện cộng song song (theo nhiều luồng) các phần tử của mảng với một hằng số *k* (xem giáo trình).

220

## III. MÔ HÌNH LẬP TRÌNH

### Assignment: Parallel Programming with PCN

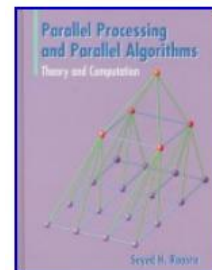
Seyd H. Roosta, Parallel Processing and Parallel Algorithms, Pages 149-164



<http://books.google.com/books?id=pi6dl5eWJpkC>

### Assignment: Parallel Programming with UNIX

Seyd H. Roosta, Parallel Processing and Parallel Algorithms, Pages 140-149



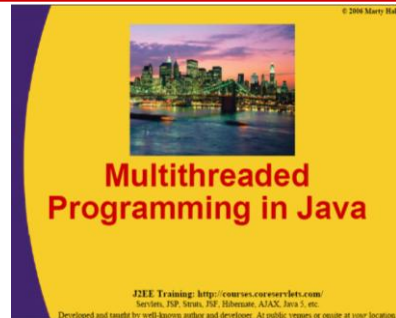
<http://books.google.com/books?id=pi6dl5eWJpkC>

222

**Assignment: POSIX Threads Programming****POSIX Threads Program****Table of Contents**

1. Abstract
2. Pthreads Overview
1. What is a Thread?
2. What are Pthreads?
3. Why Pthreads?
4. Designing Threaded Programs
3. The Pthreads API
4. Creating Threaded Programs
5. Thread Management
1. Creating and Terminating Threads
2. Passing Arguments to Threads
3. Joining and Detaching Threads
4. Stack Management
5. Miscellaneous Routines
6. Mutex Variables
1. Mutex Variables Overview
2. Creating and Destroying Mutexes
3. Locking and Unlocking Mutexes
7. Condition Variables
1. Condition Variables Overview
2. Creating and Destroying Condition Variables
3. Waiting and Signaling on Condition Variables
8. LINT: Static Information and Recommendations
9. Topics Not Covered
10. Pthread Library Routines Reference

- <https://computing.llnl.gov/tutorials/pthreads/>
- <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>
- <http://www.humanfactor.com/pthreads/>

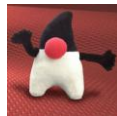
**Assignment: Parallel Programming with Thread in JAVA**

<http://java.sun.com/docs/books/tutorial/essential/threads/>  
<http://courses.coreservlets.com>

224

**1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG****Thread in JAVA**

*Black coffee for today -  
- Thread*



<http://java.sun.com/docs/books/tutorial/essential/threads/>

225

**1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG****Thread in JAVA**

- Là cơ chế điều khiển bằng NNLT để thể hiện sự song song trong một chương trình.
- Cho phép cài đặt các chương trình thực hiện nhiều tác vụ cùng một lúc (ví dụ?)
- Luồng cần có thể được...
  - Tạo lập (tăng thêm đơn vị song song)
  - Đồng bộ hóa (điều phối)
  - Hủy bỏ (giảm bớt đơn vị song song)
- Trong Java, luồng là một loại "đối tượng hệ thống":
  - Các phương thức trên đối tượng luồng
  - Mỗi đối tượng là một đơn vị song song có thể được thực hiện một cách độc lập

226

**1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG****Lợi ích?**

- Ứng dụng máy khách/đơn người dùng:
  - Tải dữ liệu / trang Web từ mạng
  - Đáp ứng sự kiện GUI trong lúc xử lý IO / Database
  - Tăng tốc xử lý khi có nhiều bộ xử lý
- Ứng dụng máy chủ:
  - Đa kết nối
- Khó gỡ rối và bảo trì, giới hạn tính khả chuyển (portable)

227

**1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG****Xử lý luồng trong Java**

- Java là ngôn ngữ lập trình hướng đối tượng hỗ trợ đa luồng, tiện lợi cho các ứng dụng web.
- Trong mô hình hướng đối tượng, tiến trình và thủ tục là thứ yếu, mọi chức năng của chương trình được xác định thông qua các đối tượng.
- Cũng giống như tiến trình, luồng được tạo lập, sau đó thực hiện một số công việc và kết thúc hoạt động khi không còn vai trò sử dụng.

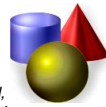
228

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG

### Hai hướng tiếp cận Threads trong JAVA

#### 1. Xây dựng lớp con của lớp Thread.

(Trong Java có một lớp đã được xây dựng sẵn là *Thread*, Sử dụng *Thread* làm lớp cơ sở để xây dựng những lớp kế thừa (lớp con) mới.



#### 2. Cài đặt giao diện Runnable.

See also:

<http://java.sun.com/docs/books/tutorial/essential/threads/>



229

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG

### b. Tạo lập các luồng

```
class MyClass extends Thread{
    // Một số thuộc tính
    public void run(){
        // Các lệnh cần thực hiện theo luồng
    }
    // Một số phương thức khác được viết đè hay bổ sung
}
```

- Khi chương trình chạy nó sẽ gọi một phương thức đặc biệt đã được khai báo trong *Thread* đó là *start()* để bắt đầu một luồng đã được tạo ra.

230

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG

### b. Tạo lập các luồng

- Java không hỗ trợ đa kế thừa. Do đó, nếu người lập trình muốn tạo ra một lớp kế thừa từ một lớp cơ sở và để thực hiện được theo luồng thì nó cũng đồng thời phải kế thừa từ lớp *Thread*. Điều này không thực hiện được. Java giải quyết hạn chế trên bằng cách xây dựng khái niệm *Interface*. Giao diện hỗ trợ thực hiện theo luồng là *Runnable*. Người lập trình thiết kế các lớp thực hiện theo luồng bằng cách cài đặt theo giao diện *Runnable*.

```
class MyClass implements Runnable{
```

```
...
}
```

231

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG

### c. Các trạng thái của Thread

Một luồng có thể ở một trong các trạng thái sau:

**new:** khi một luồng mới được tạo ra với toán tử *new()*.

**ready:** khi chúng ta gọi phương thức *start()* để bắt đầu của một luồng.

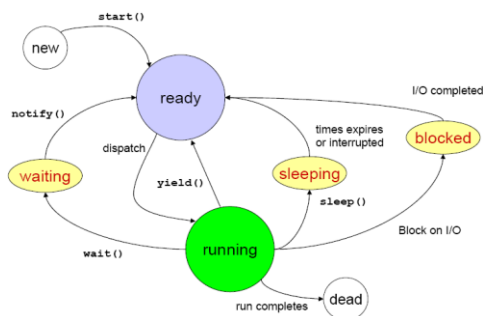
**blocked:** từ trạng thái *runnable* chuyển sang trạng thái “bị chặn” khi gọi một trong các phương thức: *sleep()*, *suspend()*, *wait()*, hay bị chặn lại ở *Input/output*.

**dead:** luồng chuyển sang trạng thái “chết” khi nó kết thúc hoạt động bình thường, hoặc gặp phải ngoại lệ không thực hiện tiếp được.

Hình dưới đây mô tả sơ đồ chuyển trạng của các luồng trong hệ thống.

232

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG



233

## 1.2 LẬP TRÌNH CHIA SẺ BỘ NHỚ DỰA VÀO LUỒNG

- Phương thức *yield()* sẽ ngưng luồng hiện hành để cho một luồng khác có cùng độ ưu tiên chạy

- wait()*: giống *yield()*, nhưng yêu cầu một luồng khác phải đánh thức nó

```
while (!condition) wait();
```

- notify()*: Luồng nào có thể ảnh hưởng đến *condition* sẽ gọi *notify()* để phục hồi luồng đang chờ

- Lập trình viên phải chịu trách nhiệm bảo đảm mỗi *wait()* có một *notify()* tương ứng

234

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

### d. Điều gì xảy ra khi có Luồng mới?

- Luồng chính vẫn tiếp tục
- Luồng mới thì hành phương thức run() và "kết thúc" khi phương thức kết thúc.
- Nếu có bất kỳ luồng nào gọi System.exit(0) thì nó sẽ "giải phóng" tất cả mọi luồng.
- Có thể xem run() như là phương thức chính của riêng mỗi luồng

235

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

### e. Chấm dứt một Luồng

- Luồng kết thúc khi phương thức run() kết thúc
- Tuy nhiên, điều gì xảy ra khi luồng đang "ngủ" (sleeping) hoặc đang bị "khóa" (blocked)?
- Đây là lúc mà vai trò của phương thức interrupt() được thể hiện. Khi interrupt() được gọi trên một Luồng đang bị "khóa", luồng sẽ bị chấm dứt.

236

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

### f. Các vấn đề khác về Luồng

- Chia sẻ và đồng bộ hóa
  - Các luồng có thể cùng truy cập đến các đối tượng được kết hợp với một tiến trình đơn.
- Lập lịch
  - Nếu (# luồng) != (# vi xử lý), thì việc lập lịch cho các luồng là một vấn đề
  - Các thao tác của các luồng khác nhau có thể xảy ra theo thứ tự bất kỳ
  - Các luồng có thể được thiết lập độ ưu tiên

237

## 1.2 LẬP TRÌNH CHIA SẼ BỘ NHỚ DỰA VÀO LUỒNG

Ví dụ: sử dụng Thread trong JAVA để tính tổng các phần tử của một mảng.

[Xem giáo trình](#)

238

## 2. TÍNH TOÁN PHÂN TÁN

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

### Ưu điểm:

- Cho phép chia sẻ dữ liệu được lưu ở nhiều máy tính khác nhau (không có bộ nhớ chia sẻ).
- Chia sẻ với nhau về một số chức năng chính của máy tính.
- Độ tin cậy và khả năng thứ lỗi cao hơn. Trong trường hợp có một máy tính bị sự cố thì những máy tính khác có thể thay thế để hoàn thành nhiệm vụ của hệ thống.

### Nhược điểm:

Những vấn đề liên quan đến việc *quản trị hệ thống, định vị tài nguyên, vấn đề đảm bảo an toàn, an ninh thông tin*, v.v. không bằng hệ tập trung

239

## 2. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### 4.2.1 Mô hình truyền thông điệp (Message Passing)

- Các đơn vị XLSS trong mô hình truyền thông điệp là các *tiến trình*.
- Các tiến trình có thể thực hiện trên những bộ xử lý khác nhau và không truy cập được vào không gian địa chỉ chia sẻ.
- Chỉ có kênh truyền là có thể chia sẻ cho các tiến trình, thường đó là LAN hoặc mạng điện rộng. 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, DCE, v.v.
- Trong các hệ thống phân tán không có bộ nhớ chia sẻ để trao đổi dữ liệu với nhau việc trao đổi được thực hiện bằng cách truyền thông điệp. Mô hình Client-Server cũng có thể sử dụng cơ chế này để cài đặt.

240



## II. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### 1. Mô hình truyền thông điệp (Message Passing)

- 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 biến 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 **không** xuất hiện trong mô hình này.
- Việc đồng bộ hoá các tiến trình của một chương trình song song cũng được thực hiện theo có chế truyền thông điệp. Nghĩa là, khi *một tiến trình muốn gửi một thông điệp thì nó phải chờ cho đến khi tiến trình nhận sẵn sàng nhận thông điệp đó* và ngược lại, cũng tương tự. Ở đây *không có các buffer để tạm lưu giữ các thông điệp cần trao đổi giữa các tiến trình*.

241

## II. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### 2. Mô hình tổng quát

- Trong mô hình truyền thông điệp, các tiến trình chia sẻ với nhau *kênh truyền thông*.
- 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 và có một tiến trình khác yêu cầu nhận (*receive*) thông điệp đó.
- Sự trao đổi được hoàn tất khi dữ liệu đã được chuyển từ nơi gửi tới nơi nhận.

242

## II. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### 2. Mô hình tổng quát

Có hai loại mô hình truyền thông điệp: *dị bộ và đồng bộ*

#### a. Truyền thông điệp dị bộ:

- Trong mô hình này, *một kênh truyền được giả thiết là có khả năng tiếp nhận không bị giới hạn* (bằng cách sử dụng buffer) để tiếp nhận các thông điệp gửi đến cho mỗi tiến trình.
- Do đó, tiến trình gửi sẽ không phải chờ để tiến trình nhận sẵn sàng nhận mà cứ gửi khi có yêu cầu.
- Hai tiến trình gửi và nhận có thể hoạt động gần như độc lập 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.
- Tiến trình nhận thì phải chờ cho đến khi có được thông điệp của một tiến trình khác gửi cho nó.

243

## II. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### Một số vấn đề nảy sinh trong dị bộ

- Khi tiến trình A gửi một thông điệp cho tiến trình B thì sau đó A cần phải 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 không nhận được câu trả lời thì nó cũng không có cách nào biết được là thông điệp đó đã được gửi đến đích chưa hoặc tiến trình B bị huỷ bỏ trong quá trình xử lý hoặc ngay cả khi câu trả lời của B không đến được A.
- 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 hoặc chương trình sẽ không thực hiện được hoặc phương thức gửi bị chặn lại.

*Điều này vi phạm ngữ nghĩa của mô hình truyền thông điệp dị bộ.*

244

## II. TÍNH TOÁN PHÂN TÁN: MÔ HÌNH TRUYỀN THÔNG ĐIỆN

### b. Truyền thông điệp 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. Mô hình này có thể tìm thấy ở những hệ thống đa bộ xử lý được cài đặt dựa trên các Transputer.
- Hệ thống truyền thông điệp đồng bộ hoàn toàn giống như hệ điện thoại, kênh truyền bị chặn lại trong quá trình đàm thoại. Hệ truyền dị bộ lại giống nhiều hơn với hệ thống bưu chính, người nhận phải chờ cho đến khi có thư được gửi đến.
- Hầu hết các thư viện lập trình hỗ trợ mô hình truyền thông điệp dị bộ.

245

## III. MÔ HÌNH LẬP TRÌNH

*(Trong phần này chúng ta định nghĩa mô hình lập trình cho hệ thống truyền thông điệp dị bộ.)*

- Lập trình theo mô hình truyền thông điệp trong hệ thống nhiều máy tính có thể thực hiện theo ba cách:
  - Sử dụng NNLT song song đặc biệt, ví dụ Occam được thiết kế để sử dụng với các Transputer (Inmos 1986)
  - Sử dụng NNLT bậc cao (tuần tự) truyền thông được mở rộng bằng cách bổ sung thêm các từ khoá và mở rộng cú pháp để xử lý việc trao đổi thông điệp, ví dụ CC++ (mở rộng của C++)
  - Sử dụng những NNLT bậc cao và cung cấp hệ thống chương trình thư viện gồm những thủ tục xử lý việc trao đổi thông điệp, ví dụ ngôn ngữ C và hệ chương trình thư viện để chạy với PVM.

*Chúng ta tập trung nghiên cứu cách thứ ba.*

246

### III. MÔ HÌNH LẬP TRÌNH

#### 1. Phát sinh các tiến trình con (1/5)

##### Mục đích:

- Để thực hiện những công việc con của một chương trình song song.
- Việc phát sinh tiến trình con thường xảy ra khi 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**.

247

### III. MÔ HÌNH LẬP TRÌNH

#### 1. Phát sinh các tiến trình con (2/5)

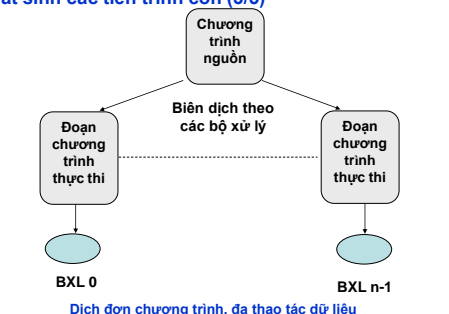
##### a. Tạo lập tiến trình tĩnh:

- Tất cả các tiến trình phải được xác định trước khi thực hiện
- Trong hệ thống có một số tiến trình được xác định trước:
  - một tiến trình điều khiển còn được gọi là **tiến trình chủ** (*master*),
  - những tiến trình khác được gọi là **tiến trình tớ** (*slave*).
- 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 sau:

248

### III. MÔ HÌNH LẬP TRÌNH

#### 1. Phát sinh các tiến trình con (3/5)



Hệ thư viện MPI được xây dựng theo cách tạo lập tĩnh như trên

249

### III. MÔ HÌNH LẬP TRÌNH

#### 1. Phát sinh các tiến trình con (4/5)

##### b. Tạo lập tiến trình động:

- Các tiến trình được tạo lập sau đó chúng có thể thực hiện trong các tiến trình khác.
- Các tiến trình có thể được tạo lập mới, bị hủy bỏ có điều kiện hoặc một số tiến trình có thể thay đổi trong quá trình thực hiện.
- Kiến trúc cho phép thực hiện tạo lập tiến trình động là MPMD (MIMD), trong đó những chương trình khác nhau có thể thực hiện trên những bộ xử lý khác nhau.

250

### III. MÔ HÌNH LẬP TRÌNH

#### 1. Phát sinh các tiến trình con (5/5)

Tất cả các hệ thư viện truyền thông điệp đều có hàm phát sinh các tiến trình con với cấu trúc tổng quát như sau:

`spawn(prog: string, host: int, count: int, id[:int])`

Trong đó,

- **prog**: tên chương trình được nạp xuống như là chương trình con. Kết quả khi thực hiện thành công sẽ trả lại mảng **id** là địa chỉ của các tiến trình được tạo ra để truyền thông điệp. Mảng **id** có chỉ số từ 1, tiến trình đầu tiên của chương trình song song có chỉ số là 0.
- **host**: tên của tiến trình mà ở đó các tiến trình con được tạo ra.
- **count**: số các đại diện của chương trình được tạo ra.

251

### III. MÔ HÌNH LẬP TRÌNH

Tiểu luận: tìm hiểu về MPI:

<http://www-unix.mcs.anl.gov/mpi/>

<http://www.mpi-forum.org/>

[http://en.wikipedia.org/wiki/Message\\_Passing\\_Interface](http://en.wikipedia.org/wiki/Message_Passing_Interface)

Quick Access

- [MPI Forum](#)
- [MPI Standard 1.1](#)
- [MPI Standard 2.0](#)
- [MPICH Home Page](#)
- [ASL/ASU MPI implementation](#)
- [Download MPICH](#)
- [Free implementation of MPI](#)

## The Message Passing Interface (MPI) standard



### What is MPI?

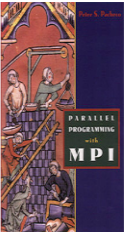
MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users.

- The [MPI standard](#) is available.
- MPI was designed for high performance on both massively parallel machines and on workstation clusters.
- MPI is widely available, with both free available and vendor-supplied [implementations](#).
- MPI was developed by a broadly based [committee](#) of vendors, implementors, and users.
- Information for [implementors of MPI](#) is available.
- [Test Suites](#) for MPI implementations are available.

### How can I learn about MPI?

### ASSIGNMENT: Parallel Programming with MPI

<http://www.cs.usfca.edu/mpl/> <http://www.mcs.anl.gov/learning.html>  
<http://www-unix.mcs.anl.gov/mpl/> <http://www.mpi-forum.org/>



**Parallel Programming with MPI**  
by  
Peter Pacheco

You can download source code for all the programs in the book. The code is available in either C (updated files were created using the Unix utilities tar and compress. If you're having trouble unpacking them, you can find the C code in the book) or Fortran (updated 2002/10/16) and Notes (updated 2000/01/30) will be put online as soon as they become available.

**Implementations of MPI**

There are a number of freely available implementations of MPI that run on a variety of platforms:

- The MPICH implementation runs on a wide range of platforms and operating systems including Unix.
- The LAM implementation runs on networks of Unix/Posix workstations.
- MP-MPICH runs on Unix systems, Windows NT and Windows 2000/XP Professional.
- The WPMPI implementation runs on Windows platforms running Windows 95/98, ME, NT and 2000.
- MacMPI is a partial implementation of MPI for Macintosh computers.

*Parallel Programming with MPI* is an elementary introduction to programming parallel systems that use the MPI 1.1 library of extensions to C and Fortran. It is intended for use by students and professionals with some knowledge of programming conventional, single-processor systems, but who have little or no experience programming multiprocessor systems. It is an extensive revision and expansion of *A User's Guide to MPI*.

### III. MÔ HÌNH LẬP TRÌNH

#### 2. Ngữ nghĩa của kênh truyền (1/2)

- Một số hệ thống đưa ra khái niệm thông điệp có cấu trúc. Các kênh được định nghĩa theo các kiểu của thông điệp và một số kênh đặc biệt chỉ truyền tải được một số kiểu nhất định.
  - Một tiến trình muốn gia nhập vào một kênh có kiểu xác định thì phải tự gắn bó với kênh đó.
  - Một kênh cũng có thể phục vụ cho một nhóm các tiến trình trong một chương trình song song.
  - Các kênh được mô tả như sau:
- Khai báo một kênh được gắn với kiểu**  
`channel mych(id1:type1, id2:type2, ..., idn:typen)`  
 Kênh mych chuyển tải được những thông điệp được xác định như trong các đối số.
  - Gắn tiến trình với kênh truyền**  
`join_channel(mych: string);`  
 Gọi ở tiến trình hiện thời để gia nhập vào kênh mych.

255

### III. MÔ HÌNH LẬP TRÌNH

#### 2. Ngữ nghĩa của kênh truyền (1/2)

- Những hệ thống khác nhau sẽ cung cấp những kênh truyền ở những mức trừu tượng khác nhau.
- Hầu hết các hệ thống đều đảm bảo rằng các kênh truyền thông không có các lỗi thông thường. Nghĩa là, tất cả các thông điệp được gửi đi và sẽ đến được đích mà không bị sai lạc.
- Kênh truyền thông có thể được xem như là hàng đợi, và thông điệp sẽ được gửi đến đích theo thứ tự nó được gửi đi.
- Kênh truyền thông cũng có thể được xem như là hộp thư (mailbox) của một tiến trình nhận/gửi thông điệp. Ở mức trừu tượng này sẽ loại bỏ được ràng buộc về thứ tự đến của các thông điệp.
- Tiến trình có thể chọn các thông điệp từ hộp thư theo một số tiêu chí nào đó.

254

### III. MÔ HÌNH LẬP TRÌNH

#### 3. 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.
- Việc lựa chọn thông điệp để nhận có thể dựa vào tiến trình gửi, kênh truyền, hay thẻ bài (*tag*) của thông điệp, v.v.
- Lời gọi `send()` là không bị chặn, không phải chờ câu trả lời của nơi nhận. Nhưng lời gọi `receive()` sẽ bị chặn lại, chỉ có tiếp tục khi có một thông điệp tương ứng đã được gửi đi.

256

### III. MÔ HÌNH LẬP TRÌNH

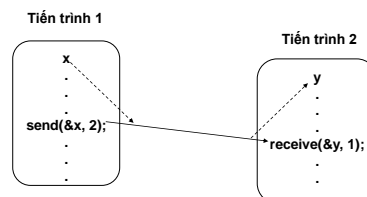
#### 3. Các hàm `send()` và `receive()`

- Gửi thông điệp cho một tiến trình id:**  
`send(id: int, message: message_type);`  
`send(id: int, tag: int, message: message_type);`
- Gửi thông điệp tới một kênh:** một thông điệp có thể gửi cho tất cả các tiến trình trên cùng một kênh mych.  
`send(mych: channel, message: message_type);`
- 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);`
- 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);`

257

### III. MÔ HÌNH LẬP TRÌNH

Ví dụ: trong C chúng ta có thể gọi  
`send(&x, destination_id);` {ở tiến trình nguồn}  
`receive(&y, source_id);` {ở tiến trình đích}  
 để gửi giá trị dữ liệu x từ tiến trình nguồn (*source-id*) sang biến y cho tiến trình đích.



Sự trao đổi thông điệp giữa hai tiến trình

258

### III. MÔ HÌNH LẬP TRÌNH

#### 4. Truy vấn trên kênh

**Mục đích:** kiểm tra thông điệp gửi đi có đến đích hay không?

Để giải quyết vấn đề này, hầu hết các chương trình thư viện cung cấp các hàm truy vấn để biết các trạng thái của kênh truyền.

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

`empty(ch: channel);`

2. Hàm gọi để xác định xem thông điệp đang chờ để nhận có phải được gửi từ tiến trình id và có thể tag?

`probe(id: int, tag: int);`

259

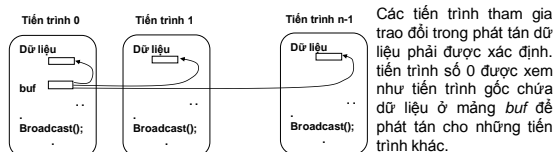
### III. MÔ HÌNH LẬP TRÌNH

#### 5. Truyền thông theo nhóm

**Mục đích:** Nhiều chương trình 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.

Các hàm để thực hiện truyền thông theo nhóm:

`Broadcast(mych:channel,tag:int, msg:message_type);` phát tán cùng một thông điệp cho tất cả các tiến trình trên kênh *mych*.



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()`.

260

### III. MÔ HÌNH LẬP TRÌNH

#### 5. Truyền thông theo nhóm

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

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

Trong đó,

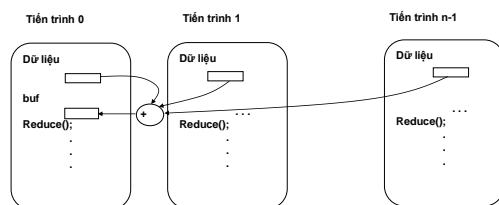
$Op\_type = \{MAX, MIN, SUM, PROD, LAND, LOR, BAND, BOR, LXOR, BXOR\}$

261

### III. MÔ HÌNH LẬP TRÌNH

#### 5. Truyền thông theo nhóm

**Ví dụ:** sơ đồ dưới đây mô tả hàm `Reduce()` tập hợp các giá trị từ *n* tiến trình và thực hiện phép cộng (`SUM`) ở tiến trình gốc.



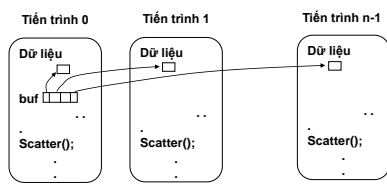
262

### III. MÔ HÌNH LẬP TRÌNH

`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*.



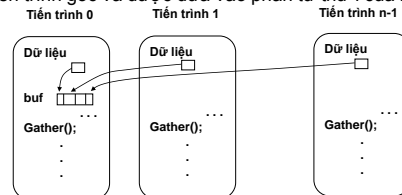
263

### III. MÔ HÌNH LẬP TRÌNH

`Gather()`: 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,sendbuff[N]:DataType);`

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*,



264

### III. MÔ HÌNH LẬP TRÌNH

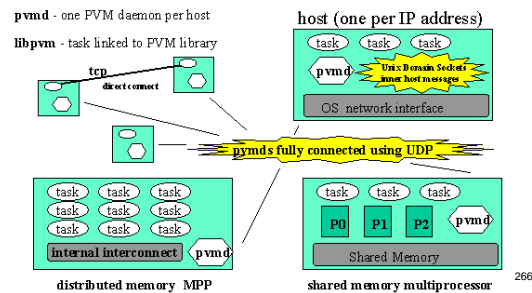
**Barrier():** thực hiện việc đồng bộ hoá những tiến trình cùng gia nhập một kênh truyền. Mỗi tiến trình phải chờ cho đến khi tất cả các tiến trình khác trên kênh đạt đến điểm đồng bộ hoá bằng lời gọi Barrier() trong chương trình.

**Barrier(mych:channel);**

265

### IV. LẬP TRÌNH TRÊN CỤM MÁY

#### How PVM is Designed



266

### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

Nhận xét 1:

- Mô hình truyền thông điệp được sử dụng rất hiệu quả để lập trình song song theo cụm máy tính.
- PVM cung cấp môi trường phần mềm để truyền thông điệp cho các cụm máy tính thuần nhất và cả không thuần nhất.
- PVM có một tập hợp các hàm thư viện được viết bằng C hoặc Fortran.
- Mỗi chương trình được viết bằng C, được biên dịch để chạy trên những kiểu máy tính xác định trên mạng.
- Mỗi nút trong mạng (LAN) phải truy cập đến hệ thống tệp chứa các chương trình đã được dịch để thực hiện.

267

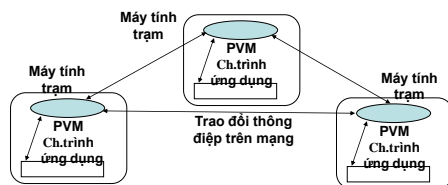
### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

Nhận xét 2:

- Cụm máy tính trong PVM phải được xác định theo các mức ưu tiên để thực hiện các chương trình.
- 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. Các tiến trình PVM yêu cầu đủ thông tin để chọn lựa đường truyền dữ liệu.

268

### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH



Sự trao đổi thông điệp của các máy tính trong hệ PVM

269

### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

- 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.

**Một số hàm thông dụng trong PVM:**

- pvm\_spawn():** phát sinh tiến trình mới trong PVM
- pvm\_myid():** ghi danh tiến trình muốn tham gia vào hệ PVM
- pvm\_exit():** huỷ bỏ tiến trình
- pvm\_send() và pvm\_recv():** Các chương trình trao đổi thông điệp với nhau.

270

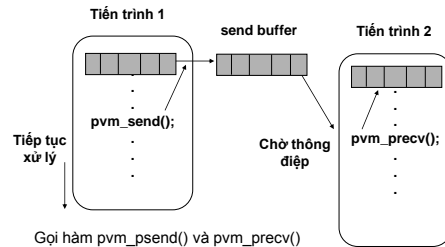
#### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

- 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_prekv()`.

271

#### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

Hình dưới đây mô tả hoạt động của hai tiến trình trao đổi một mảng dữ liệu với nhau.



272

#### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

- Khi dữ liệu chuyển 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()` dùng cho dữ liệu kiểu int, `pvm_pkfloat()` và `pvm_upkfloat()` dùng cho dữ liệu kiểu float, `pvm_pkstr()` và `pvm_upkstr()` dùng 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_initend()`.

273

#### IV. LẬP TRÌNH TRÊN CỤM MÁY TÍNH

Tương tự, các lệnh khác về trao đổi thông điệp theo nhóm như:

- `pvm_bcast()`
- `pvm_scatter()`
- `pvm_gather()`
- `pvm_reduce()`, v.v.

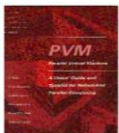
Xem ví dụ ở tài liệu

274

Tiểu luận: tìm hiểu về PVM

<http://www.csm.ornl.gov/pvm/manpages.html>

<http://www.netlib.org/pvm3/>



PVM: Parallel Virtual Machine  
A User's Guide and Tutorial for Networked Parallel Computing

- Al Geist
- Adam Hegeman
- Jack Dongarra
- Weicheng Jiang
- Robert Manjesh
- Yaeli Sanderam

[pvm@netlib.org](mailto:pvm@netlib.org)

275

#### V. ĐÁNH GIÁ CHƯƠNG TRÌNH SONG SONG

Thời gian thực hiện song song  $t_p = t_{comp} + t_{comm}$

Trong đó,

$t_{comp}$  : thời gian tính toán ;  $t_{comm}$  : thời gian truyền thông.

- 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.
- Trong phân tích độ phức tạp tính toán, chúng ta luôn giả thiết rằng, tất cả các bộ xử lý là giống nhau và thao tác cùng một tốc độ như nhau. Đối với những cụm máy tính không thuần nhất thì điều này không đảm bảo. Do đó, việc đánh giá thời gian tính toán của những hệ như thế là khó.

276

## V. ĐÁNH GIÁ CHƯƠNG TRÌNH SONG SONG

Ví dụ:

Giả sử cần cộng  $n$  số trên hai máy tính, trong đó mỗi máy cộng  $n/2$  số với nhau và lưu ở máy tính của mình.

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

277

### Bài tập

4.1 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?

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

279

### Bài tập

4.7 Viết chương trình song song theo luồng trong mô hình chia sẻ bộ nhớ để thực hiện nhân hai ma trận cấp  $n \times n$ .

4.8 Xây dựng hệ xử lý hình ống để tính  $\sin x$

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

4.9 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)
```

281

## V. ĐÁNH GIÁ CHƯƠNG TRÌNH SONG SONG

Bài toán được phát biểu như sau:

1. Máy tính 1 gửi  $n/2$  số cho máy tính 2
2. Cả hai máy tính cộng  $n/2$  số một cách đồng thời
3. Máy tính 2 chuyển kết quả tính được về máy tính 1
4. Máy tính 1 cộng hai kết quả để có kết quả cuối cùng.

Thời gian tính toán (ở bước 2 và 4):  $t_{\text{comp}} = n/2 + 1$

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

$$\begin{aligned} t_{\text{comm}} &= (t_{\text{startup}} + n/2 * t_{\text{data}}) + (t_{\text{startup}} + t_{\text{data}}) \\ &= 2 * t_{\text{startup}} + (n/2 + 1) * t_{\text{data}} \end{aligned}$$

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

278

### Bài tập

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

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

4.4 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.

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

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

280

### Bài tập

4.10 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.

282

## CHƯƠNG 5. THUẬT TOÁN SONG SONG

## NỘI DUNG

1. Nguyên lý thiết kế thuật toán song song
2. Các cách tiếp cận trong thiết kế TTSS
3. Phân tích và đánh giá thuật toán song song
4. Các thuật toán sắp xếp, tìm kiếm song song
5. Một số thuật toán song song thông dụng.

## 5.1 NGUYÊN LÝ THIẾT KẾ THUẬT TOÁN SONG SONG

## 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*.

Tổng quát, *thuật toán song song* là một tập các tiến trình (process) hoặc các tác vụ (task) có thể thực hiện đồng thời và có thể trao đổi dữ liệu với nhau để kết hợp cùng giải một bài toán đặt ra.

284

## 5.1 NGUYÊN LÝ THIẾT KẾ THUẬT TOÁN SONG SONG

## 05 câu hỏi đặt ra trước khi thiết kế một 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?

285

## 5.1 NGUYÊN LÝ THIẾT KẾ THUẬT TOÁN SONG SONG

## Các nguyên lý cơ bản trong thiết kế TTSS (1/2)

## 1. Nguyên lý lập lịch:

Giảm tối thiểu các bộ xử lý sử dụng trong thuật toán sao cho thời gian tính toán là không tăng (xét theo khía cạnh độ phức tạp).

Nghĩa là, nếu độ phức tạp tính toán của thuật toán là  $O(f(n))$  thì thời gian thực hiện của chương trình có thể tăng khi số bộ xử lý giảm, và thời gian tính toán tổng thể tăng lên một hằng số nào đó - nhưng vẫn là  $O(f(n))$ .

2. Nguyên lý hình ống: Nguyên lý này được áp dụng khi bài toán xuất hiện một dãy các thao tác  $\{T_1, T_2, \dots, T_n\}$ , trong đó  $T_{i+1}$  thực hiện sau khi  $T_i$  kết thúc.

286

## 5.1 NGUYÊN LÝ THIẾT KẾ THUẬT TOÁN SONG SONG

## Các nguyên lý cơ bản trong thiết kế TTSS (2/2)

## 3. Nguyên lý chia để trị:

Chia bài toán thành những phần nhỏ hơn tương đối độc lập với nhau và giải quyết chúng một cách song song.

## 4. Nguyên lý đồ thị phụ thuộc dữ liệu:

Phân tích mối quan hệ dữ liệu trong tính toán để xây dựng đồ thị phụ thuộc dữ liệu và dựa vào đó để xây dựng thuật toán song song.

## 5. Nguyên lý điều kiện tương tranh:

Nếu hai tiến trình cùng muốn truy cập vào cùng một mục dữ liệu chia sẻ thì chúng phải tương tranh với nhau, nghĩa là chúng có thể cản trở lẫn nhau.

287

## 5.1 NGUYÊN LÝ THIẾT KẾ THUẬT TOÁN SONG SONG

## Ngoài ra khi thiết kế TTSS cần phải quan tâm

- Cấu hình topology liên kết mạng: cũng một thuật toán song song cài đặt trên hai máy tính có cấu hình topology liên kết mạng khác nhau thì có thể có độ phức tạp khác nhau.

Ví dụ: DAP là máy tính kiểu SIMD với  $64 \times 64$  bộ xử lý, thời gian nhân ma trận là tuyến tính theo kích cỡ của ma trận và phụ thuộc vào đường truyền dữ liệu giữa các hàng với cột.

- Nhiều thuật toán song song được thiết kế dựa trên những kiến thức về kiến trúc máy tính, ngôn ngữ lập trình song song và các phương pháp tính toán.

288



## 5.2 CÁC CÁCH TIẾP CẬN TRONG THIẾT KẾ TTSS

### Có ba cách tiếp cận để 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 topology và môi trường song song thực tế.

Nhận xét: Cách 1 và 2 là thông dụng nhất

289

## 5.2 CÁC CÁCH TIẾP CẬN TRONG THIẾT KẾ TTSS

### Chú ý:

**khi chuyển một thuật toán tuần tự sang thuật toán song song hoặc chuyển một TTSS thích hợp với kiến trúc đang có.**

### Cần trả lời hai câu hỏi:

1. Kiến trúc nào phù hợp cho bài toán?
2. Những bài toán loại nào sẽ xử lý hiệu quả trong kiến trúc song song cho trước?

290

## 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

### Độ phức tạp:

- Thời gian
- Không gian

### Độ phức tạp:

- Tuần tự
- Song song

291

## 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

### Tuần tự?

**Định nghĩa:** Một thuật toán có độ phức tạp tính toán  $f(x) = O(g(x)) \Leftrightarrow \exists C \geq 0, x_0 \in \mathbb{N}$  sao cho  $0 \leq f(x) \leq C \cdot g(x)$ , với mọi số lượng dữ liệu vào  $x \geq x_0$ .

Ví dụ: cho  $f(n) = 1+2+\dots+n$   
 Vì  $f(n) < n + n + \dots + n = n^2$  nên  $f(n)$  là  $O(n^2)$  với  $C=x_0=1$

### Song song?

Độ phức tạp tính toán của TTSS không chỉ phụ thuộc vào kích cỡ của dữ liệu đầu vào mà còn phụ thuộc vào kiến trúc máy tính song song và số lượng các bộ xử lý được phép sử dụng trong hệ thống.

292

## 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

### Song song?

*Độ phức tạp theo thời gian của TTSS sử dụng  $p$  bộ xử lý để giải một bài toán có kích cỡ  $n$  là hàm  $f(n,p)$  xác định thời gian cực đại giữa thời điểm bắt đầu thực hiện thuật toán bởi một bộ xử lý và thời điểm kết thúc của các bộ xử lý đối với bộ dữ liệu vào bất kỳ.*

Có hai loại phép toán khác nhau trong các TTSS:

- Các phép toán cơ sở như  $+$ ,  $-$ ,  $*$ ,  $/$ , AND, OR, v.v.
- Các phép toán truyền dữ liệu trên các kênh truyền.

293

## 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

### Các định nghĩa

**Định nghĩa 1** (định lý Brent):

Một thuật toán song song có độ phức tạp tính toán  $O(T)$  với  $P$  bộ xử lý khi nó thực hiện nhiều nhất là  $O(T \cdot P)$  phép toán cơ sở.

(giới hạn số lượng phép toán cơ sở được thực hiện của một thuật toán có độ phức tạp cho trước)

294

### 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

#### Các định nghĩa

##### Định nghĩa 2:

Một thuật toán song song có độ phức tạp tính toán  $O(T)$  sử dụng nhiều bộ xử lý để thực hiện  $O(e)$  phép toán cơ sở thì khi cài đặt với  $P$  bộ xử lý sẽ có độ phức tạp thời gian là  $O((e/P) + T)$ .

Định nghĩa 2 chỉ ra rằng khi số bộ xử lý được sử dụng giảm xuống trong một phạm vi nhất định thì thuật toán tiếp tục làm việc nhưng thời gian thực hiện sẽ tăng lên.

295

### 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

#### Các định nghĩa

##### Định nghĩa 3:

Một thuật toán song song có độ phức tạp tính toán  $O(T)$  với  $P$  bộ xử lý có thể cài đặt với  $[P/p]$ ,  $1 \leq p \leq P$  bộ xử lý thì sẽ có độ phức tạp thời gian là  $O(p \cdot T)$ .

Định nghĩa 3 khẳng định rằng có cách để cài đặt thuật toán song song khi số các bộ xử lý được sử dụng bị giảm xuống.

296

### 5.3 PHÂN TÍCH VÀ ĐÁNH GIÁ THUẬT TOÁN SONG SONG

#### • **Mức độ song song của thuật toán:**

là số lượng cực đại các phép toán độc lập có thể thực hiện đồng thời ở mỗi thời điểm thực hiện của thuật toán.

• **Hệ số gia tốc: để biết mức độ song song hóa của TTSS**  
Hệ số gia tốc của TTSS sử dụng  $p$  bộ xử lý được xác định:

$$S_p = \frac{T_s}{T_p}$$

Trong đó,

+  $T_s$  là thời gian thực hiện tính toán trên một bộ xử lý

+  $T_p$  là thời gian thực hiện tính toán trên  $p$  bộ xử lý.

Với giả thiết là bộ xử lý tuần tự và các bộ xử lý song song là như nhau.

297

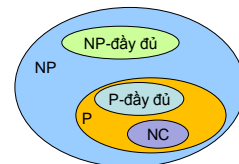
### 5.4 NHỮNG BÀI TOÁN GIẢI ĐƯỢC TRÊN MÔ HÌNH PRAM

- $P$ : lớp các bài toán có thể giải được trong thời gian đa thức
- $NP$  là lớp các bài toán, mà mọi nghiệm giả định đều có thể được kiểm chứng trong thời gian đa thức.
- Hiển nhiên  $P \subseteq NP$ .
- Gọi  $C$  là bài toán khó nhất trong  $NP$ .  $C$  được gọi là  $NP$ -đầy đủ.

Nếu  $C \in P$  thì kết luận  $P=NP$

Nếu  $C \notin P$  thì kết luận  $P \subset NP$

**$P \neq NP$  ?**



298

### 5.4 NHỮNG BÀI TOÁN GIẢI ĐƯỢC TRÊN MÔ HÌNH PRAM

## Nếu $P = NP$ ?

- Mọi bài toán kiểm chứng được thì cũng giải được dễ dàng
- Các bài toán tối ưu tổ hợp đều giải được trong thời gian đa thức.
- Mọi lo lắng về sự bùng nổ tổ hợp không còn phải quan tâm
- Một loạt các hệ mã khóa công khai dựa trên giả thiết  $P \neq NP$  sẽ bị đổ vỡ.

299

### 5.4 NHỮNG BÀI TOÁN GIẢI ĐƯỢC TRÊN MÔ HÌNH PRAM

Giả sử  $T(n)$  là hàm xác định trên biến nguyên  $n$ . Ta ký hiệu:

+  $T(n)^{O(1)}$  - tập các hàm lũy thừa của  $T(n)$ .

+  $(\log n)^{O(1)}$  - tập các hàm lũy thừa của logarit.

Ví dụ:  $\log^2 n, \log^5 n \in \log^{O(1)}$ ,  $n^2, n^5 \in n^{O(1)}$  và  $e^{2n}, e^{5n} \in (e^n)^{O(1)}$ .

**Mệnh đề 5.1** (định đề tính toán song song): Lớp các bài toán giải được trong thời gian  $T(n)^{O(1)}$  bởi mô hình PRAM bằng lớp tất cả các bài toán giải được trong không gian  $T(n)^{O(1)}$  bởi mô hình RAM (máy tính tuần tự với bộ nhớ ngẫu nhiên) nếu  $T(n) \geq \log n$ .

Từ định đề này suy ra: PRAM có thể giải được những bài toán **NP-đầy đủ** trong thời gian đa thức.

See also: pages 45-48 Parallel computing by Michael J. Quinn

300

## 5.5 CÁC CẤU TRÚC LỆNH SONG SONG

### 1. Khối các lệnh song song:

Parbegin và Parend (Cobegin và Coend): Giả sử các lệnh  $S_1, S_2, \dots, S_n$  được thực hiện trên  $n$  tiến trình riêng biệt

Parbegin	Cobegin
$S_1;$	$S_1;$
$S_2;$	$S_2;$
...	...
$S_n;$	$S_n;$
Parend	Coend

301

## 5.5 CÁC CẤU TRÚC LỆNH SONG SONG

### 2. Cấu trúc forall (parfor)

Nhiều khi một số tiến trình (câu lệnh) tương tự nhau cần phải bắt đầu thực hiện và cùng lặp lại một số lần. Điều này có thể thực hiện được bằng cấu trúc *forall*

For (i=0; i<n; i++) {	Hoặc	Forall (i=0; i<n; i++) {
$S_1;$		$S_1;$
$S_2;$		$S_2;$
...		...
$S_n;$		$S_n;$
}		}
In parallel		Parend

302

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### 1. Thuật toán sắp xếp theo hạng

**Bài toán:** cho mảng  $a[n]$  các số nguyên. Hãy sắp xếp tăng dần các phần tử của mảng.

**Ý tưởng thuật toán:** 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. Giả sử có  $n$  số được lưu giữ trong mảng  $a[n]$  và kết quả sau khi sắp xếp là  $b[n]$ .

**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];
}
```

Độ phức tạp  $O(n^2)$

303

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### Thuật toán song song-Sử dụng $n$ bộ xử lý

Giả sử ta có  $n$  bộ xử lý. Mỗi bộ xử lý được cấp một số trong dãy số cho trước và nó phải tìm vị trí của số đó trong dãy được sắp xếp.

```
for (i=0; i<n; i++) { // n bộ xử lý thực hiện song song
    x = 0;
    for (j=0; j<n; j++)
        if (a[i]>a[j]) x++; // đếm số phần tử nhỏ hơn
    b[x] = a[i]; // Sao sang bảng được sắp xếp
}
```

Nhận xét: Tất cả  $n$  bộ xử lý thực hiện song song nên thuật toán có độ phức tạp là  $O(n)$  (tuyến tính).

(Xem tài liệu *Thuật toán song song-Sử dụng  $n^2$  bộ xử lý*)

304

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### 2. Thuật toán sắp xếp so sánh và đổi chỗ

**Bài toán:** cho mảng  $a[n]$  các số nguyên. Hãy sắp xếp tăng dần các phần tử của mảng.

**Ý tưởng thuật toán:** so sánh hai phần tử liền kề nhau. Nếu chưa theo thứ tự thì đổi chỗ nhau. Quá trình lặp lại cho đến khi nào không còn cặp nào không thỏa mãn thì dừng.

**Thuật toán tuần tự:**

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

305

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### 2. Thuật toán sắp xếp so sánh và đổi chỗ

**Thuật toán song song:**

**Ý tưởng thuật toán**

- Dùng  $n$  tiến trình 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:** gồm các tiến trình được đánh số chẵn so sánh với những tiến trình tiếp theo (số lẻ), nếu nó giữ phần tử lớn hơn thì trao đổi dữ liệu với tiến trình đó.
  - Pha lẻ:** các tiến trình được đánh số lẻ hoạt động tương tự như trên

306

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

Ví dụ:  $n=8$  và  $a = (4, 2, 7, 8, 5, 1, 3, 6)$

Pha/TT	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$	$P_7$
0	4	2	7	8	5	1	3	6
1	2	4	7	8	1	5	3	6
2	2	4	7	8	1	5	3	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

Sắp xếp theo nguyên lý hình ống  $\longleftrightarrow$  Đổi chỗ  
 $\dashrightarrow$  Không đổi chỗ <sup>307</sup>

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### Thuật toán song song:

Giả sử A lưu dữ liệu ở tiến trình lẻ và B lưu dữ liệu ở tiến trình chẵn  
 Thuật toán song song theo hình ống được mô tả theo MPI như sau:

#### Pha chẵn

$P_i, i = 0, 2, \dots, n-2$

`recv (&A,  $P_{i+1}$ );`

`send (&B,  $P_{i+1}$ );`

`if (A>B) B=A;`

$P_i, i = 1, 3, \dots, n-3$

`send (&A,  $P_{i-1}$ );`

`recv (&B,  $P_{i-1}$ );`

`if (A>B) A=B;`

#### Pha lẻ

$P_i, i = 1, 3, \dots, n-3$

`send (&A,  $P_{i+1}$ );`

`recv (&B,  $P_{i+1}$ );`

`if (A>B) A=B;`

$P_i, i = 0, 2, \dots, n-2$

`recv (&A,  $P_{i-1}$ );`

`send (&B,  $P_{i-1}$ );`

`if (A>B) B=A;` <sup>308</sup>

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

Kết hợp 2 pha ta được thuật toán song song:

$P_i, i = 1, 3, \dots, n-3$

`send (&A,  $P_{i-1}$ );`

`recv (&B,  $P_{i-1}$ );`

`if (A>B) A=B;`

`if (i<=n-3) {`

`send (&A,  $P_{i+1}$ );`

`recv (&B,  $P_{i+1}$ );`

`if (A>B) A=B;`

`}`

$P_i, i = 0, 2, \dots, n-2$

`recv (&A,  $P_{i+1}$ );`

`send (&B,  $P_{i+1}$ );`

`if (A>B) B=A;`

`if (i>=2) {`

`recv (&A,  $P_{i-1}$ );`

`send (&B,  $P_{i-1}$ );`

`if (A>B) B=A;`

`}`

Nhận xét: thuật toán có độ phức tạp là  $O(n)$ .

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### 3. Thuật toán sắp xếp MergeSort

**Bài toán:** cho mảng  $a[n]$  các số nguyên. Hãy sắp xếp tăng dần các phần tử của mảng.

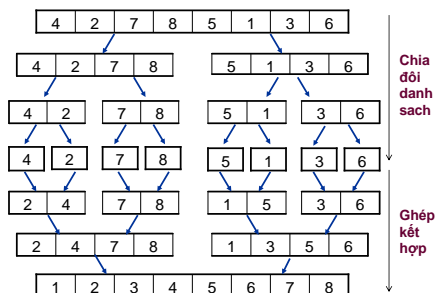
**Ý tưởng thuật toán** (dùng nguyên tắc chia để trị)

- Chia mảng  $a$  thành 2 phần
- Mỗi phần lại chia đôi tiếp cho đến khi mỗi phần nhỏ chỉ có 1 phần tử
- Từng cặp được ghép lại theo thứ tự sắp xếp

#### Ví dụ

Mảng  $a = (4, 2, 7, 8, 5, 1, 3, 6)$

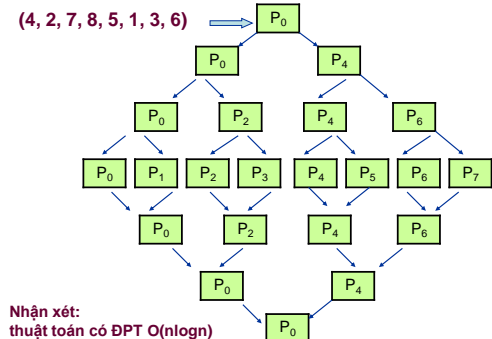
## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG



311

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

Chia danh sách thành từng cặp và phân công cho các BXL



Nhận xét:  
 thuật toán có ĐPT  $O(n \log n)$

312

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

Xác định thời gian truyền thông:  $t_{comm}$

a. Trong giai đoạn phân chia dữ liệu

Truyền dữ liệu

$t_{startup} + (n/2)t_{data}$

$t_{startup} + (n/4)t_{data}$

$t_{startup} + (n/8)t_{data}$

$t_{startup} + (n/16)t_{data}$

$t_{startup} + (n/32)t_{data}$

$t_{startup} + (n/64)t_{data}$

$t_{startup} + (n/128)t_{data}$

$t_{startup} + (n/256)t_{data}$

$t_{startup} + (n/512)t_{data}$

$t_{startup} + (n/1024)t_{data}$

$t_{startup} + (n/2048)t_{data}$

$t_{startup} + (n/4096)t_{data}$

$t_{startup} + (n/8192)t_{data}$

$t_{startup} + (n/16384)t_{data}$

$t_{startup} + (n/32768)t_{data}$

$t_{startup} + (n/65536)t_{data}$

$t_{startup} + (n/131072)t_{data}$

$t_{startup} + (n/262144)t_{data}$

$t_{startup} + (n/524288)t_{data}$

$t_{startup} + (n/1048576)t_{data}$

$t_{startup} + (n/2097152)t_{data}$

$t_{startup} + (n/4194304)t_{data}$

$t_{startup} + (n/8388608)t_{data}$

$t_{startup} + (n/16777216)t_{data}$

$t_{startup} + (n/33554432)t_{data}$

$t_{startup} + (n/67108864)t_{data}$

$t_{startup} + (n/134217728)t_{data}$

$t_{startup} + (n/268435456)t_{data}$

$t_{startup} + (n/536870912)t_{data}$

$t_{startup} + (n/1073741824)t_{data}$

$t_{startup} + (n/2147483648)t_{data}$

$t_{startup} + (n/4294967296)t_{data}$

$t_{startup} + (n/8589934592)t_{data}$

$t_{startup} + (n/17179869184)t_{data}$

$t_{startup} + (n/34359738368)t_{data}$

$t_{startup} + (n/68719476736)t_{data}$

$t_{startup} + (n/137438953472)t_{data}$

$t_{startup} + (n/274877906944)t_{data}$

$t_{startup} + (n/549755813888)t_{data}$

$t_{startup} + (n/1099511627776)t_{data}$

$t_{startup} + (n/2199023255552)t_{data}$

$t_{startup} + (n/4398046511104)t_{data}$

$t_{startup} + (n/8796093022208)t_{data}$

$t_{startup} + (n/17592186044416)t_{data}$

$t_{startup} + (n/35184372088832)t_{data}$

$t_{startup} + (n/70368744177664)t_{data}$

$t_{startup} + (n/140737488355328)t_{data}$

$t_{startup} + (n/281474976710656)t_{data}$

$t_{startup} + (n/562949953421312)t_{data}$

$t_{startup} + (n/1125899906842624)t_{data}$

$t_{startup} + (n/2251799813685248)t_{data}$

$t_{startup} + (n/4503599627370496)t_{data}$

$t_{startup} + (n/9007199254740992)t_{data}$

$t_{startup} + (n/18014398509481984)t_{data}$

$t_{startup} + (n/36028797018963968)t_{data}$

$t_{startup} + (n/72057594037927936)t_{data}$

$t_{startup} + (n/144115188075855872)t_{data}$

$t_{startup} + (n/288230376151711744)t_{data}$

$t_{startup} + (n/576460752303423488)t_{data}$

$t_{startup} + (n/1152921504606846976)t_{data}$

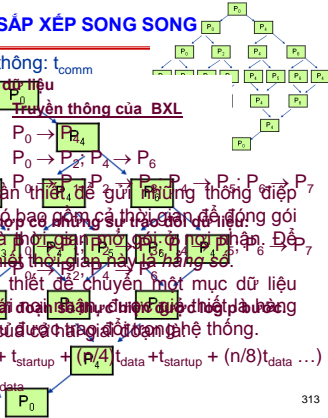
$t_{startup} + (n/2305843009213693952)t_{data}$

$t_{startup} + (n/4611686018427387904)t_{data}$

$t_{startup} + (n/9223372036854775808)t_{data}$

$t_{startup} + (n/18446744073709551616)t_{data}$

$t_{startup} + (n/36893488147419103232)t_{data}$



## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### e. Song song hóa QuickSort

#### Phát biểu lại QuickSort

- Chọn phần tử *pilot* để chia dãy thành hai phần: bên trái là những phần tử nhỏ hơn hoặc bằng *pilot* và bên phải là những phần tử lớn hơn *pilot*.
- Phần tử *pilot* được chèn vào giữa hai dãy con được tách và nó là vị trí đã được sắp xếp sau khi thực hiện bước 1.
- Bước 1 lặp lại một cách đệ quy cho đến khi các dãy con chỉ còn lại một phần tử.

319

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

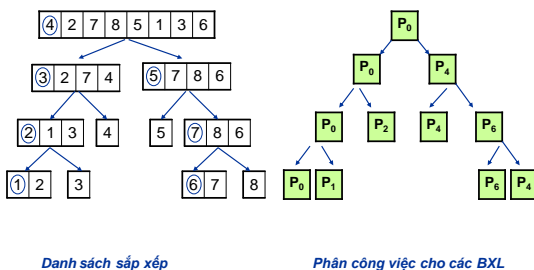
### Song song hóa QuickSort

**Ý tưởng:** bắt đầu thực hiện ở một BXL và chuyển các lời gọi đệ quy cho các BXL khác.

- Tạo ra tập các tiến trình và đặt dãy cần sắp xếp vào *stack*.
- Tiến trình đầu tiên (tiến trình chủ) thực hiện tách dãy con trước thành hai dãy con và đặt vào *stack*.
- Những tiến trình khác (tiến trình tớ) xử lý những dãy con đã được tách ra và thực hiện những công việc tương tự như thuật toán đã nêu.

320

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG



321

## 5.6 CÁC THUẬT TOÁN SẮP XẾP SONG SONG

### f. Đánh giá thuật toán QuickSort song song

- Giả thiết rằng *pilot* được chọn một cách lý tưởng để mỗi lần đều tách thành hai phần có kích cỡ bằng nhau.

#### 1. Thời gian tính toán.

- Đầu tiên một bộ xử lý phải thao tác với  $n$  số. Sau đó hai bộ xử lý thao tác trên  $n/2$  số, rồi bốn bộ xử lý thao tác trên  $n/4$  số, v.v.
- Thời gian để thực hiện thuật toán sẽ là:  

$$t_{\text{comp}} = n + n/2 + n/4 + \dots \approx 2n$$

#### 2. Thời gian truyền thông.

- Sự trao đổi dữ liệu giữa các bộ xử lý chỉ xảy ra khi thực hiện ghép kết hợp giống như MergeSort.
- $$t_{\text{comm}} = 2(t_{\text{startup}} + (n/2)t_{\text{data}} + t_{\text{startup}} + (n/4)t_{\text{data}} + t_{\text{startup}} + (n/8)t_{\text{data}} \dots)$$

$$= 2(\log p)t_{\text{startup}} + 2nt_{\text{data}}$$

322

## 5.7 CÁC THUẬT TOÁN SONG SONG THÔNG DỤNG

- Nhân ma trận
  - Mô hình lưới 2 chiều
  - Mô hình mảng tam thu
- Giải hệ phương trình tuyến tính
- Tính biểu đồ của ảnh
- Tô màu đồ thị
- Thuật toán Kruskal tìm cây khung cực tiểu
- Thuật toán tính tổng tiền tố
- Phương trình nhiệt một chiều

Bài tập (tham khảo tài liệu)

323