BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
---------o0o---------

TÍNH TOÁN PHÂN TÁN VÀ ỨNG DỤNG

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
NGÀNH CÔNG NGHỆ THÔNG TIN

Sinh viên thực hiên: Nguyễn Thị Phương
Giáo viên hướng dẫn: Ths. Nguyễn Trịnh Đông
Mã số sinh viên: 110942

MỤC LỤC

MỤC LỤC 1

DANH MỤC HÌNH VẼ 5

DANH MỤC CHỮ VIẾT TẮT 6

MỞ ĐẦU 7

CHƯƠNG I: MÁY TÍNH SONG SONG 8

1.1. Giới thiệu chung về tính toán song song và phân tán 8

1.2. Trình bày về máy tính song song 8

1.2.1. Kiến trúc và các loại máy tính song song 8

1.2.2. Các thành phần của máy tính song song 12

1.2.3. Chương trình dịch và các hệ điều hành 15

1.3. Kỹ thuật lập trình song song 16

1.3.1. Những mô hình lập trình song song 16

1.3.2. Nguyên lý thiết kế thuật toán song song 24

1.4. Một số chiến lược song song hóa phổ biến 25

1.4.1. Song song hóa kết quả 26

1.4.2. Song song hóa đại diện 26

1.4.3. Song song hóa chuyên biệt 27

CHƯƠNG II : MÁY ẢO SONG SONG PVM (Paralle Virtual Machine) 28

2.1. Giới thiệu chung 28

2.1.1. Máy tính xử lý song song MPP 28

2.1.2. Máy trạm thay thế (Cluster of Workstation) 28

2.1.3. Tính toán trên mạng không đồng nhất 29

2.2. Kiến trúc của máy ảo song song PVM (Parallel Virtual Machine) 29

2.2.1. Định nghĩa 29

2.2.2. Nguyên lý của một hệ thống PVM 29

2.2.3. Cấu trúc của PVM 30

2.2.4. Kiến trúc của PVM 31
2.3. Cơ chế hoạt động 31
2.4. Lập trình trên cụm máy tính PVM 32
2.4.1. Điều khiển các task 34
2.4.2. Truyền thông điệp 34
2.4.3. Nhận thông điệp 35
2.4.4. Nhóm các task 35
2.4.5. Các kiểu dữ liệu được đóng gói trong PVM 36
2.5. Sử dụng PVM 36
2.5.1. Cài đặt PVM 36
2.5.2. Bắt đầu với PVM 37
2.5.3. Một số vấn đề khi sử dụng PVM 38
2.5.4. Chạy chương trình PVM 39
2.5.5. Giao diện điều khiển PVM 40
2.5.6. Các tùy chọn của hostfile 41
2.6. Lập trình dùng PVM 43
2.6.1. Mô hình Master – Slave 43
2.6.2. Mô hình Task – to – Task 44
2.7. Thiết kế môi trường hỗ trợ tính toán song song 47
2.7.1. Quản lý biến phân chia 48
2.7.2. Giao diện với người lập trình 54
CHƯƠNG 3: THỰC NGHIỆM 55
3.1. Phát biểu bài toán 55
3.2. Xây dựng các toán tử trong bài toán 55
3.3. Cài đặt và đánh giá 58
3.3.1. Cấu hình hệ thống 58
3.3.2. Cài đặt PVM 59

3.3.3. Biên dịch và chạy thử 59
3.3.4. Kết quả.....60
3.3.5. Đánh giá.....61
KẾT LUẬN 62
Tài liệu tham khảo 63

DANH MỤC HÌNH VẼ

Hình 1.1. Hệ thống bộ nhớ phân cấp.

Hình 1.2. Mô hình bộ nhớ kết hợp.

Hình 1.3. Mô hình mạng liên kết n bộ xử lý.

Hình 1.4. Mô hình chia sẻ bộ nhớ.

Hình 1.5. Mô hình bộ nhớ phân tán.

Hình 1.6. Mô hình "đường - ống".

Hình 1.7. Dịch đơn chương trình, đa thao tác dữ liệu.

Hình 2.1. Kiến trúc của PVM.

Hình 2.2. Sự trao đổi thông điệp của các máy tính trong hệ PVM.

Hình 2.3. Gọi hàm pvm_psend() và pvm_precv().

Hình 2.4. Phương thức trao đổi các tiến trình.

Hình 2.5. Mô tả các giai đoạn của quá trình biên dịch.

Hình 2.6. Mô hình quản lý biến phân chia.

Hình 2.7. Mô hình quản lý tiến trình.

Hình 2.8. Mô hình cơ chế hoạt động.

Hình 2.9. Quản lý biến phân chia.

Hình 2.10. Giao thức truyền thông.

Hình 3.1: Node Slave đã mount được thư mục /home của node Master.

Hình 3.2. PVM đã được cài và đã add host Slave.

Hình 3.3. Kết quả của bài toán.

DANH MỤC CHỮ VIẾT TẮT

ALU	Arithmetic and Logic Unit
AM	Associative Memory
BXL	Bộ xử lý
COMA	Cache Only Memory Architecture
CPU	Computer Processing Unit
HĐH	Hệ điều hành
UMA	Uniform Memory Access
MISD	Multiple Instruction Stream, Single Data Stream
MPSD	Multiple Program Stream, Single Data Stream
MIMD	Multiple Instruction Stream, Multiple Data Stream
MPMD	Multiple Program Stream, Multiple Data Stream
MPI	Message Passing Interface
MPP	Machine Massively Parallel Processors
NUMA	Non – Uniform Memory Access
PRAM	Parallel Random Access Memory
PVM	Parallel Virtual Machine
SISD	Single Instruction Stream, Single Data Stream
SPSD	Single Program Stream, Single Data Stream
SIMD	Single Instruction Stream, Multiple Data Stream
RAM	Random Access Memory

MỞ ĐẦU

Nhiệm vụ của Công nghệ thông tin là nghiên cứu các mô hình, phương pháp và công nghệ, công cụ hỗ trợ để tạo ra những hệ thống phần mềm giải quyết được những bài toán phức tạp của thực tế. Những vấn đề về xử lý ngôn ngữ tự nhiên, tiếng nói, nhận dạng dự báo thời tiết,…đều đòi hỏi phải xử lý dữ liệu với tốc độ rất cao, với khối lượng dữ liệu rất lớn. Hầu hết những bài toán này, những máy tính xử lý tuần tự kiểu Von Neumann như hiện nay là không đáp ứng yêu cầu. Mặc dù tốc độ và số lượng các bộ xử lý tăng nhiều trong những năm qua, nhưng do giới hạn về phương diện vật lý nên khả năng tính toán của chúng không thể tăng mãi theo yêu cầu hiện tại, càng không đáp ứng trong tương lai. Điều này dẫn tới là muốn tăng được khả năng tính toán của các hệ thống máy tính để giải được những bài toán đáp ứng yêu cầu thực tế thì không còn cách nào khác là phải khai thác được những khả năng xử lý song song và phân tán của hệ thống máy tính hiện đại.

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

Các công cụ hỗ trợ lập trình song song có thể kể đến như MPI, PVM, một số được tích hợp sẵn thành chuẩn trong các ngôn ngữ lập trình như thư viện OpenMP trong C/C++, Fortran,…Trong khuôn khổ bài khóa luận em sẽ tìm hiểu về lập trình song song sử dụng PVM, cấu hình PVM và chạy một ví dụ ứng dụng. Nội dung của bài khóa luận bao gồm:

Chương 1: Tìm hiểu về máy tính song song.

Chương 2: Tìm hiểu về máy ảo song song PVM.

Chương 3: Thực nghiệm và chạy ứng dụng.

Kết luận: Nêu lên những vấn đề đã nghiên cứu được và những hạn chế, thiếu sót và phương hướng phát triển trong tương lai.

CHƯƠNG I: MÁY TÍNH SONG SONG

1.1. Giới thiệu chung về tính toán song song và phân tán

Xử lý song song là quá trình xử lý gồm nhiều tiến trình được kích hoạt đồng thời và cùng tham gia giải quyết một vấn đề, nói chung là thực hiện trên những hệ thống có nhiều bộ xử lý đồng thời.

1.2. Trình bày về máy tính song song

1.2.1. Kiến trúc và các loại máy tính song song

a. Tại sao phải xử lý song song

- Phân biệt xử lý song song với tuần tự:

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

Mục đích của xử lý song song: là thực hiện tính toán nhanh trên cơ sở sử dụng nhiều BXL đồng thời. Cùng với tốc độ xử lý nhanh hơn, việc xử lý song song cũng sẽ giải được những bài toán phức tạp yêu cầu khối lượng tính toán lớn.

- Ba yếu tố chính dẫn đến việc xây dựng các hệ thống xử lý song song:
  - Tốc độ xử lý của các BXL theo kiểu von Neumann đã dần tiến tới giới hạn, không thể cải tiến thêm được do vậy dẫn tới đòi hỏi phải thực hiện xử lý song song.
  - Hiện nay giá thành của phần cứng (CPU) giảm mạnh, tạo điều kiện để xây dựng những hệ thống có nhiều BXL với giá thành hợp lý.
  - Sự phát triển của công nghệ mạch tích hợp cho phép tạo ra những hệ thống có hàng triệu transistor trên một chip.

Vấn đề xử lý song song liên quan trực tiếp đến: kiến trúc máy tính, phần mềm hệ thống (hệ điều hành), thuật toán và ngôn ngữ lập trình, v.v.

Hệ thống tính song song: là một tập các BXL (thường là cùng một loại) kết nối với nhau theo một kiến trúc nào đó để có thể hợp tác với nhau trong hoạt động và trao đổi dữ liệu được với nhau.

Chúng ta dễ nhận thấy là độ phức tạp của xử lý song song sẽ lớn hơn xử lý tuần tự rất nhiều, và tập trung chủ yếu ở phương diện trao đổi dữ liệu và đồng bộ các tiến trình.

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

b. Phân loại kiến trúc máy tính

Dựa vào các đặc tính về số lượng BXL, số chương trình thực hiện, cấu trúc bộ nhớ, v.v., Michael Flynn (1966) đã phân máy tính thành 4 loại sau:

- Mô hình SISD(Single Instruction Stream, Single Data Stream): Đơn luồng lệnh, đơn luồng dữ liệu.

Máy tính loại SISD chỉ có một CPU, ở mỗi thời điểm thực hiện một chỉ lệnh và chỉ đọc, ghi một mục dữ liệu. Tất cả các máy tính SISD chỉ có một thanh ghi register được gọi là bộ đếm chương trình (program counter) được sử dụng để nạp địa chỉ của lệnh tiếp theo và kết quả là thực hiện theo một thứ tự xác định của các câu lệnh.

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

- Mô hình SIMD (Simple Instruction Stream Multiple Data Stream): Đơn luồng lệnh, đa luồng dữ liệu.

Máy tính loại SIMD có một đơn vị điều khiển để điều khiển nhiều đơn vị xử lý (nhiều hơn một đơn vị) thực hiện theo một luồng các câu lệnh. CU phát sinh tín hiệu điều khiển tới tất cả các phần tử xử lý, những BXL này cùng thực hiện một phép toán trên các mục dữ liệu khác nhau, nghĩa là mỗi BXL có luồng dữ liệu riêng.

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

- Mô hình MISD (Multiple Instruction Simple Data): Đa chỉ lệnh, đơn dữ liệu.

Máy tính loại MISD là ngược lại với SIMD. Máy tính MISD có thể thực hiện nhiều chương trình (nhiều lệnh) trên cùng một mục dữ liệu, nên còn được gọi là MPSD (đa chương trình, đơn dữ liệu).

Kiến trúc kiểu này có thể chia thành hai nhóm:

- Lớp các máy tính gồm nhiều đơn vị xử lý (PU) khác nhau có thể nhận được những chỉ lệnh khác nhau để thực hiện trên cùng một mục dữ liệu. Đây là kiến trúc khó và hiện nay chưa có loại máy tính nào được sản xuất theo loại này.
- Lớp các máy tính có các luồng dữ liệu được gửi tuần tự theo dãy các CPU liên tiếp. Đây là loại kiến trúc hình ống, xem xét như sau:

Nguyên lý hình ống (pipelined) dựa vào phương pháp phân đoạn hoặc chia nhỏ một tiến trình tính toán thành một số đoạn nhỏ hơn để thực hiện trong các pha liên tiếp. Tất cả các giai đoạn của một tiến trình được thực hiện tuần tự, khi thực hiện xong thì bắt đầu thực hiện của tiến trình tiếp theo. Mỗi pha thực hiện xong sẽ gửi kết quả cho pha tiếp theo. Như vậy, trong cách thực hiện theo nguyên lý hình ống, khi một giai đoạn công việc đang thực hiện thì một giai đoạn khác có thể nạp dữ liệu vào, và dữ liệu vào của giai đoạn này có thể là kết quả của giai đoạn trước nó.

- Mô hình MIMD (Multiple Instruction Multiple Data): Đa luồng lệnh, đa luồng dữ liệu.

Máy tính loại MIMD còn gọi là đa BXL, trong đó mỗi BXL có thể thực hiện những luồng lệnh (chương trình) khác nhau trên các luồng dữ liệu riêng. Hầu hết các hệ thống MIMD đều có bộ nhớ riêng và cũng có thể truy cập vào được bộ nhớ chung (global) khi cần, do vậy giảm thiểu được thời gian trao đổi dữ liệu giữa các BXL trong hệ thống. Đây là kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất và đã có nhiều máy tính được sản xuất theo kiến trúc này, ví dụ: BBN Butterfly, Alliant FX, iSPC của Intel, v.v

c. Kiến trúc máy tính song song

Theo sự phân loại của Flynn thì có 2 họ kiến trúc quan trọng cho các máy tính song song đó là SIMD và MIMD. Những kiến trúc khác có thể xếp theo 2 mẫu đó.

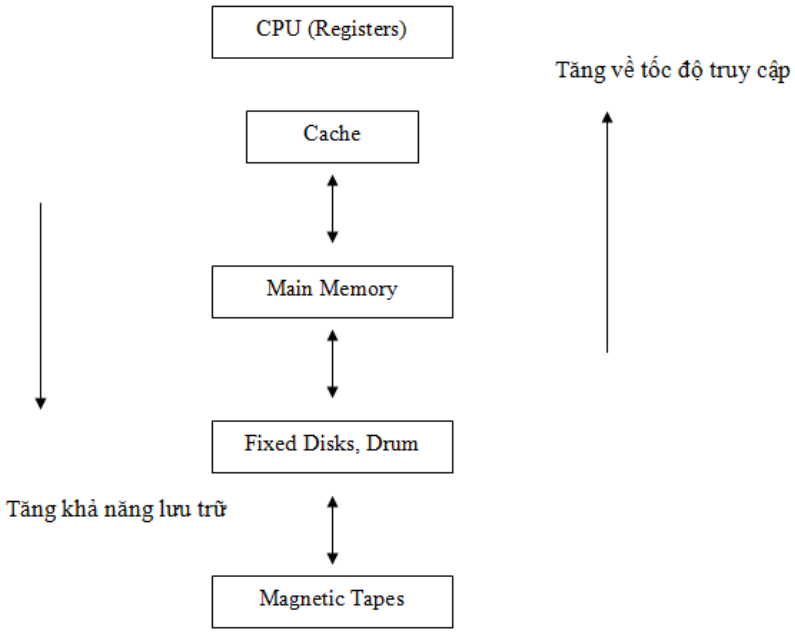
Những kiến trúc khác nhau có thể tạo ra những khả năng khác nhau cho việc xử lý song song. Đối với những kiến trúc máy tính song song thì mục đích chính là khai thác triệt để khả năng của kiến trúc song song để xây dựng chương trình song song.

d. Song song hóa máy tính tuần tự

Các hệ thống bộ nhớ phân cấp:

Tốc độ thực hiện các phép toán trong BXL nhanh hơn rất nhiều so với việc truy cập vào bộ nhớ; Tốc độ truy cập vào bộ nhớ trong (RAM) nhanh hơn rất nhiều so với việc truy cập vào bộ nhớ ngoài.

Hệ thống bộ nhớ phân cấp như thế có thể mô tả như hình 1.1.



Hình 1.1. Hệ thống bộ nhớ phân cấp

Các thanh ghi được sử dụng trực tiếp cho ALU. Bộ nhớ cache được xem như vùng đệm giữa BXL và bộ nhớ chính. Sự song song hóa trong trao đổi dữ liệu theo cấu trúc phân cấp là cách khai thác chung để cải tiến hiệu quả xử lý của hệ thống. Ví dụ, trong khi dữ liệu được lấy từ bộ nhớ ngoài vào bộ nhớ chính thì đồng thời có thể gửi dữ liệu từ cache vào cho CPU.

Đa chương trình và chia sẻ thời gian.

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

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

phần tính toán trong quá trình xử lý. Các hệ điều hành đa chương trình luân phiên thực hiện các chương trình khác nhau.

Để thực hiện việc này HĐH sử dụng Bộ lập lịch chia sẻ thời gian làm nhiệm vụ phân chia CPU cho mỗi tiến trình một khoảng thời gian cố định theo phương pháp quay vòng tròn. Bằng cách đó, tất cả các tiến trình đều được sẵn sàng để thực hiện trên cơ sở được phép sử dụng CPU và những tài nguyên khác của hệ thống. Do vậy, về nguyên tắc việc phát triển những chương trình song song trên máy đơn BXL thực hiện được nếu có hệ điều hành cho phép nhiều tiến trình thực hiện, nghĩa là có thể xem như là hệ thống đa bộ xử lý.

1.2.2. Các thành phần của máy tính song song

Xử lý song song là quá trình xử lý thông tin, trong đó các thao tác trên các phần tử dữ liệu thuộc một hoặc một số tiến trình được thực hiện đồng thời nhằm cùng giải quyết một bài toán.

1.2.2.1. Bộ nhớ

Một trong các thành phần quan trọng nhất của kiến trúc máy tính là bộ nhớ.

Một số mô hình bộ nhớ của máy tính song song.

a. Bộ nhớ kết hợp

Bộ nhớ kết hợp (AM – Associative Memory) bao gồm các ô nhớ (cell) và logic kết hợp. Mỗi ô nhớ của AM có 4 đầu vào và 2 đầu ra:

Hinh 1.2. Mô hình bộ nhớ kết hợp

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

- Hai kết quả ở đầu ra (output):
  - Bit đối sánh m chỉ ra dữ liệu được lưu trong bộ nhớ có sánh được với bit đối số a.
  - Bit kết quả q.

Tất cả các ô nhớ AM được tổ chức thành các từ (word) và được xây dựng thành mảng các ô giống nhau.

So sánh với bộ nhớ truy cập ngẫu nhiên RAM thì AM đắt hơn nhưng tốc độ tìm kiếm nhanh hơn.

b. Mô hình bộ nhớ truy cập ngẫu nhiên song song

Mô hình tính toán song song được biết dưới tên gọi PRAM bao gồm bộ nhớ chung RAM với m vùng bộ nhớ đủ lớn để chia sẻ cho p bộ xử lý.

Bộ nhớ chung được sử dụng để lưu trữ dữ liệu và là vùng để trao đổi giữa các bộ xử lý. Nó cho phép các bộ xử lý truy cập vào bộ nhớ đồng thời.

Mô hình loại này có 5 dạng sau:

- Các phương thức truy cập bộ nhớ (Access Memory Primitives).
- Mô hình UMA (Uniform Memory Access) của bộ nhớ chia sẻ.
- Mô hình NUMA (Non - Uniform Memory Access) của bộ nhớ chia sẻ.
- Kiến trúc bộ nhớ Cache – Only (COMA).
- Bộ nhớ đa máy tính.

1.2.2.2. Mạng kết nối các thành phần của máy tính song song

Trong hầu hết các kiến trúc song song thì vấn đề quan trọng nhất trong thiết kế là xác định sự liên kết giữa các bộ xử lý và bộ nhớ với nhau.

Một kiến trúc lý tưởng là kiến trúc trong đó mỗi bộ xử lý đều kết nối được với các bộ xử lý còn lại.

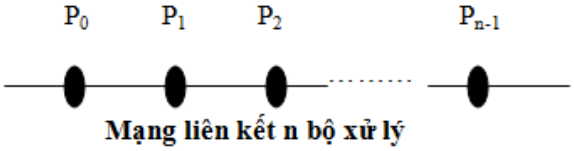
Có 2 loại cấu hình topo cho mạng liên kết:

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

Một số loại cấu hình topo của mạng liên kết giữa các bộ xử lý của máy tính song song:

a. Liên kết tuyến tính và vòng xuyến

- Trong mạng liên kết tuyến tính các bộ xử lý được tổ chức liên kết với nhau theo dãy và được đánh số theo thứ tự tăng dần:



Hình 1.3. Mô hình mạng liên kết n bộ xử lý

Mặc dù đây là mạng liên kết đơn giản nhưng dữ liệu cũng cần phải chuyển qua nhiều bộ xử lý, kết quả là sự truyền thông dữ liệu giữa các bộ xử lý, đặc biệt là giữa bộ xử lý đầu và cuối sẽ bị chậm lại khi số bộ xử lý khá hơn.

- Mạng liên kết vòng: Được tổ chức tương tự như liên kết tuyến tính nhưng bộ xử lý đầu và cuối được nối vòng với nhau.

Trong liên kết vòng, sự trao đổi giữa các bộ xử lý có thể thực hiện theo 1 chiều, gọi là mảng mạng đơn hoặc theo cả 2 chiều gọi là mạng kép. Sự truyền thông trong mạng liên kết vòng, nhất là giữa những bộ xử lý ở xa nhau thì cũng vẫn bị trễ.

b. Liên kết xáo trộn

c. Mạng liên kết lưới 2 chiều

Trong mạng liên kết mắt lưới hai chiều, mỗi bộ xử lý được liên kết với 4 láng giềng: trên, dưới, trái, phải.

Có 2 dạng :

- Lưới không quay vòng.
- Lưới quay vòng tròn.

d. Mạng liên kết siêu nối hoặc hình khối n chiều

Trong mạng liên kết hình khối, các chỉ số của các bộ xử lý được chuyển thành nhị phân và hai bộ xử lý được gọi là láng giềng với nhau nếu nhãn chỉ số của chúng sai khác nhau 1 bit.

e. Mạng liên kết hình sao

Trong mạng liên kết hình sao với n là số nguyên, bộ xử lý sẽ tương ứng với một hoán vị của n ký hiệu.

1.2.3. Chương trình dịch và các hệ điều hành

a. Chương trình dịch

Chương trình dịch được viết bằng các ngôn ngữ lập trình truyền thống thì phải được dịch sang dạng mã mà phần cứng hiểu được nó, đó là ngôn ngữ máy. Chương trình dịch và chương trình thông dịch được sử dụng để thực hiện các chuyển đổi đó.

Đối với các hệ thống song song thì một thành phần rất quan trọng là chương trình dịch song song. Chương trình dịch làm giảm được thời gian thực hiện chương trình (song song) bằng cách chia nhỏ bài toán thành các khối công việc và những khối này được xử lý đồng thời bởi nhiều đơn vị xử lý. Một số chương trình chỉ làm nhiệm vụ phát hiện những khối công việc thực hiện được song song và thực hiện phân chia các đơn vị chức năng, một số khác tinh tế hơn, có thể lập lịch cho cả bài toán.

Có 3 cách tiếp cận để xây dựng chương trình dịch cho các máy tính song song:

- *Run Time Partitioning and Run Time Scheduling:* Cách tiếp cận này phù hợp với một số ứng dụng thực tế. Tuy nhiên việc lập lịch và phân hoạch được thực hiện lúc chạy chương trình sẽ làm giảm hiệu suất của hệ thống.
- *Complie Time Partitioning and Run Time Scheduling:* Là mô hình chung để xây dựng chương trình dịch cho những đa bộ xử lý. Lập lịch phân việc được thực hiện lúc chương trình chạy, nhưng việc phân hoạch công việc thành các khối được thực hiện bởi người lập trình và chương trình dịch.
- *Complie Time Partitioning and Complie Time Scheduling:* Phân hoạch công việc và lập lịch được thực hiện ở giai đoạn dịch chương trình.

b. Hệ điều hành

Hệ điều hành là một chương trình làm nhiệm vụ phối hợp các hoạt động của máy tính.

Hệ điều hành đa bộ xử lý có nhiệm vụ tích hợp các tài nguyên tính toán và các bộ xử lý trao đổi với nhau thông qua mạng liên kết để tạo thành một hệ thống nhất làm việc cho hiệu quả.

Hệ điều hành cho các máy tính song song được phân làm 3 loại:

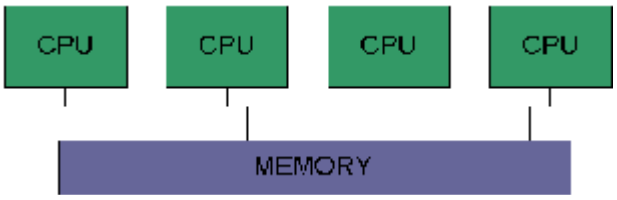
- Những hệ điều hành mở rộng và phát triển từ những hệ đơn bộ xử lý để chạy được trên những kiến trúc song song.
- Những hệ điều hành được thiết kế riêng cho những kiến trúc song song.
- Những hệ điều hành tổng hợp được thiết kế để cài đặt được trên những kiến trúc song song khác nhau.

1.3. Kỹ thuật lập trình song song

1.3.1. Những mô hình lập trình song song

1.3.1.1. Mô hình lập trình chia sẻ bộ nhớ

Mô hình chia sẻ bộ nhớ:



Hình 1.4. Mô hình chia sẻ bộ nhớ

Trong môi trường UNIX, WINDOWS chúng ta có thể tạo ra nhiều tiến trình khác nhau trong hệ thống và chúng được sử dụng để mô phỏng lập trình đa bộ xử lý.

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

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

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

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

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

Cách thức trao đổi dữ liệu giữa các tiến trình:

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

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

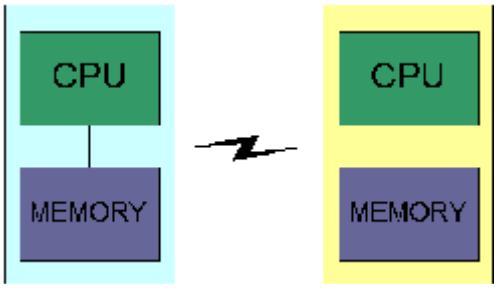
b. Lập trình chia sẻ bộ nhớ dựa vào luồng (Thread)

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

Các tiến trình và các luồng trong hệ thống song song cần phải được đồng bộ, song việc đồng bộ giữa các luồng được thực hiện hiệu quả hơn đối với các tiến trình. Đồng bộ giữa các tiến trình đòi hỏi tốn thời gian hoạt động của hệ thống,

trong khi đối với các luồng thì việc đồng bộ chủ yếu tập trung vào sự truy cập các biến chung (global) của chương trình.

1.3.1.2. Mô hình lập trình bộ nhớ phân tán (Tính toán song song phân tán)



Hình 1.5. Mô hình bộ nhớ phân tán

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

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

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

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

- Những vấn đề liên quan đến việc quản trị hệ thống, vấn đề đảm bảo an toàn hệ thống, bảo mật thông tin, v.v.
- Xử lý trong các hệ thống phân tán không có bộ nhớ chia sẻ để trao đổi dữ liệu với nhau. Sự trao đổi được thực hiện bằng cách gửi/nhận thông báo.

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

a. Mô hình gửi / nhận thông báo

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

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

Có hai mô hình gửi/nhận thông báo:

a.1. Gửi/nhận thông báo theo cơ chế dị bộ

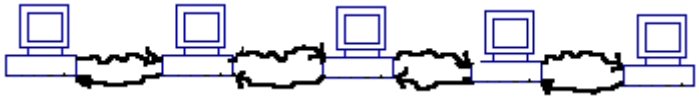
Trong mô hình này, một kênh truyền thông được giả thiết là có khả năng tiếp nhận không bị giới hạn. Khả năng không giới hạn được cài đặt trong thực tế bằng cách sử dụng bộ đệm(buffer) để tiếp nhận các thông điệp gửi đến cho mỗi tiến trình. Do vậy, tiến trình gửi sẽ không phải chờ tiến trình nhận sẵn sàng nhận mà cứ gửi khi có dữ liệu. hai tiến trình gửi và nhận có thể hoạt động gần như độc lập với nhau và thông điệp có thể nhận được sau một khoảng thời gian nào đó (lâu bất kỳ) kể từ khi nó được gửi đi. Tuy nhiên, tiến trình nhận muốn nhân dữ liệu thì phải chờ cho đến khi có thông điệp của một tiến trình khác gửi cho nó. Có một số yêu cầu sau trong truyền thông di bộ:

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

thì phương thức gửi sẽ bị chặn lại. Điều này vi phạm ngữ nghĩa của mô hình gửi/nhận thông báo dị bộ.

Các mô hình lập trình dựa trên cơ chế gửi/nhận thông báo dị bộ:

- *Mô hình hướng tâm:* Các yêu cầu và trả lời qua lại giữa khách (Client) và chủ (Server) - Đây là mô hình mà các máy tính chỉ có quan hệ gửi-nhận dữ liệu với một máy -- máy "chủ". Trong suốt quá trình tính toán, chúng không cần đến nhau.
- *Mô hình "đường-ống":* là mô hình các máy tính được hình dung là xếp thành một hàng và mỗi máy tính gửi nhận dữ liệu cho 2 máy kề bên.



Hình 1.6. Mô hình "đường - ống"

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

a.2. Gửi/nhận thông báo theo cơ chế đồng bộ

Trong mô hình này, tiến trình gửi bị chặn lại cho đến khi tiến trình nhận sẵn sàng nhận. Ở đây, sự truyền thông và đồng bộ hoá luôn gắn chặt với nhau.

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

- Ưu điểm: Làm cho nhiều vấn đề trong đồng bộ hoá và việc cấp phát bộ nhớ động trở lên đơn giản hơn.
- Nhược điểm:
  - Việc gắn chặt các tiến trình với thời gian phân phát thông điệp cũng được xem như là điều kiện ràng buộc bổ sung đòi hỏi trong khi thiết kế và thực thi chương trình.
  - Việc bắt tiến trình gửi phải chờ dẫn đến việc làm giảm tính đồng thời của hệ thống.
  - Ngoài ra, để cài đặt hiệu quả các hệ thống truyền thông đồng bộ đòi hỏi phải có những phần cứng đặc biệt để đảm bảo rằng sự truyền thông phải

cực nhanh và sự trao đổi dữ liệu không ảnh hưởng tới sự tính toán của hệ thống. Mà các mạng truyền thông nhanh có nhiều nút mạng trao đổi dữ liệu với nhau là rất đắt tiền. Vì những lý do trên, nên hệ gửi/nhận thông báo dị bộ làm việc trên mạng cục bộ đã được phát triển mạnh mẽ hơn.

b. Lập trình song song phân tán

Phần này chỉ tập trung vào cách sử dụng những ngôn ngữ lập trình bậc cao và các thư viện gồm những thủ tục xử lý việc trao đổi thông điệp, ví dụ ngôn ngữ C/C++ và hệ chương trình thư viện để chạy với PVM, MPI, …

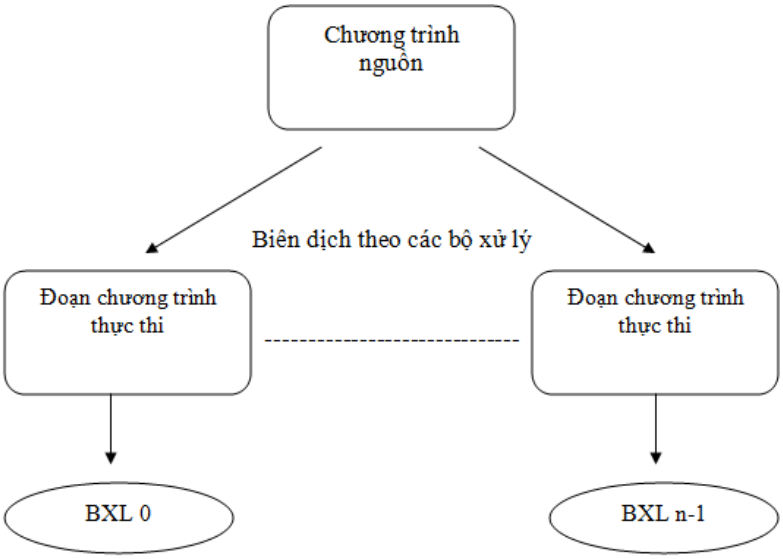
Trong hệ thống trao đổi thông điệp thì vấn đề tạo lập các tiến trình để thực hiện trên những bộ xử lý khác nhau và việc gửi, nhận thông điệp là quan trọng nhất.

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

Bước 1: Tạo các tiến trình con

Một chức năng quan trọng trong lập trình song song là tạo lập ra nhiều tiến trình để thực hiện những công việc con của một chương trình song song. Có hai cách tạo lập tiến trình:

- *Tạo lập tiến trình tĩnh:* số tiến trình được xác định trước khi thực hiện. Trong các hệ thống này thường có một tiến trình điều khiển còn được gọi là tiến trình "chủ" (master), những tiến trình khác được gọi là tiến trình tớ (slave). Đây là mô hình SPMD – sẽ có một đoạn mã chung cho tất cả các tiến trình. Sau khi chương trình nguồn được viết với các lệnh phân chia công việc cho từng tiến trình, nó sẽ được dịch sang mã thực thi được cho những tiến trình đó (Hình 1.7).
- *Tạo lập tiến trình động:* Các tiến trình có thể được tạo lập mới hoặc bị huỷ bỏ có điều kiện và số lượng tiến trình có thể thay đổi trong quá trình thực hiện. Mô hình cho phép thực hiện tạo lập động là MPMD (MIMD), trong đó những chương trình khác nhau có thể thực hiện trên những bộ xử lý khác nhau.



Hình 1.7. Dịch đơn chương trình, đa thao tác dữ liệu

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

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

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

Các dạng gửi/nhận như sau:

- 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 truyền thông: một thông điệp có thể gửi cho tất cả các tiến trình trên cùng một kênh mych (my channel):

  send(mych: channel, message: message_type);

- 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);

- Nhận từ một địa chỉ nguồn:

  receive(source_id: int, msg: 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);

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

c.1. Truy vấn trên kênh

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

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

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

  empty(ch: channel);

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

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

c.2. Truyền thông theo nhóm

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

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

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

  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().

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

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

Trong đó:

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

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

- *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*.

  Tương tự trường hợp của Broadcast(), lệnh Scatter() sẽ không thực hiện được cho đến khi tất cả các tiến trình đều thực hiện lời gọi Scatter().

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

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

  Ngược lại hàm Scatter(), dữ liệu từ tiến trình thứ i được nhận về ở tiến trình gốc và được đưa vào phần tử thứ i của mảng buf.

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

  Barrier(mych:channel);

1.3.2. Nguyên lý thiết kế thuật toán song song

Như trên đã nêu ở trên, khi xử lý song song ta phải xét cả kiến trúc máy tính và các 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.

Để thiết kế được các thuật toán song song cần phải thực hiện:

- Phân chia dữ liệu cho các tác vụ.
- Chỉ ra cách truy cập và chia sẻ dữ liệu.
- Phân các tác vụ cho các tiến trình (bộ xử lý).
- Các tiến trình được đồng bộ ra sao.

Có 5 nguyên lý chính trong thiết kế thuật toán song song:

- *Các nguyên lý lập lịch:* mục đích là 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).
- *Nguyên lý hình ống:* Nguyên lý này được áp dụng khi bài toán xuất hiện một dãy các thao tác {T1, T2, . . ., Tn}, trong đó Ti+1 thực hiện sau khi Ti kết thúc.
- *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.
- *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.
- *Nguyên lý điều kiện tranh đua:* 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.

Ngoài những nguyên lý nêu trên, khi thiết kế thuật toán song song ta còn phải chú ý đến kiến trúc của hệ thống tính toán.

1.4. Một số chiến lược song song hóa phổ biến

Trong rất nhiều chiến lược khác nhau để song song hoá thuật toán tuần tự có ba chiến lược thiết kế chương trình song song tương đối phổ biến là song song hoá kết quả, song song hoá đại diện và song song hoá chuyên biệt. Mặc dù, trong từng bài toán cụ thể việc kết hợp các chiến lược thiết kế có thể cho ta kết quả thú vị, song thông thường tuỳ thuộc tính chất bài toán chỉ một chiến lược thiết kế thuật toán song song được chọn nhằm đạt hiệu quả cao nhất.

1.4.1. Song song hóa kết quả

Việc phân loại các chiến lược thiết kế thuật toán song song phụ thuộc vào tính chất bài toán. Song song hoá kết quả là cơ chế tính toán song song tập trung trên toàn bộ dữ liệu của bài toán. Mỗi bộ xử lý sẽ cho một phần kết quả của bài toán và các bộ xử lý hoạt động song song sao cho các phần việc được thực hiện độc lập tối đa có thể. Sau khi các phần việc hoàn thành, công đoạn cuối cùng là kết hợp các thành phần để được kết quả hoàn chỉnh. Các lớp bài toán chia - để - trị thường sử dụng chiến lược song song hoá kết quả để thiết kế chương trình song song. Mỗi bài toán trong lớp các bài toán thường được chia thành các bài toán thành phần. Mỗi bài toán thành phần được giải quyết độc lập và kết quả cuối cùng là sự kết hợp các kết quả của bài toán thành phần.

Việc thiết kế chương trình theo chiến lược song song hoá kết quả thông qua năm giai đoạn :

- *Giai đoạn đầu:* cần phải mô hình kết qủa dưới dạng cấu trúc dữ liệu gồm nhiều thành phần, đồng thời xác định được sự phụ thuộc giữa chúng.
- *Giai đoạn 2:* phân mỗi bộ xử lý đảm nhiệm công việc cho một hay nhiều thành phần và việc phân chia này phải bảo đảm vấn đề hiệu suất của các bộ xử lý.
- *Giai đoạn 3:* xác định nguồn tài nguyên cần thiết để thực hiện việc tính toán các thành phần.
- *Giai đoạn 4:* xác định cách lấy các giá trị kết quả của các thành phần khi thực hiện xử lý song song.
- *Giai đoạn 5:* kết hợp các kết quả thành phần để được kết quả bài toán và kết thúc các xử lý.

1.4.2. Song song hóa đại diện

Thiết kế chương trình theo cơ chế song song hoá đại diện xác định cụ thể công việc phải thực hiện để song song hoá. Như vậy, để giải quyết một bài toán có nhiều công đoạn, mỗi công đoạn của bài toán được giải quyết song song cho đến khi hoàn thành công đoạn đó và các công đoạn tiếp theo cũng được thực hiện tương tự cho đến khi bài toán được giải quyết. Các mô hình của chiến lược này là chủ - tớ, tính toán - tổng hợp - truyền thông.

Trong mô hình chủ - tớ bài toán cần giải được chia thành các vấn đề phụ thuộc lẫn nhau. Các bộ xử lý đóng vai trò tớ trong mô hình có nhiệm vụ xử lý các

vấn đề này và giữa chúng được điều phối bởi bộ xử lý đóng vai trò chủ. Khác với phương pháp chia - để - trị, các vấn đề ở đây không nhất thiết cùng được tiến hành giải quyết, mà có thể được giải quyết song song một cách tuần tự.

Việc thiết kế chương trình theo chiến lược song song hoá đại diện được thực hiện thông qua ba giai đoạn. Giai đoạn đầu xác định các công việc cần phải thực hiện bởi các bộ xử lý. Trong giai đoạn tiếp theo cần phải xác định bộ xử lý đóng vai trò điều khiển các công việc.Cuối cùng cần phải nhận biết được kết quả công việc.

1.4.3. Song song hóa chuyên biệt

Trong chiến lược song song hoá chuyên biệt, bài toán cần giải quyết bao gồm nhiều công việc, mỗi công việc có đặc thù riêng được giao cho một bộ xử lý chuyên dụng. Ngoài ra hệ thống cần một bộ xử lý giữ vai trò agent, điều phối quá trình thực hiện công việc. Trong mỗi công việc đặc thù, các phần việc nhỏ hơn được thực hiện song song. Sau khi các phần việc này hoàn thành, cần tiến hành “phối hợp” các kết quả để hoàn thành công việc.

CHƯƠNG II : MÁY ẢO SONG SONG PVM (Paralle Virtual Machine)

Máy ảo song song là một thuật ngữ để chỉ một tập các máy tính đơn lẻ được nối kết với nhau trong cùng một mạng và cùng được sử dụng để giải quyết một bài toán.

2.1. Giới thiệu chung

2.1.1. Máy tính xử lý song song MPP

Ý tưởng chính của việc xử lý song song là chia bài toán lớn thành các công việc nhỏ hơn để xử lý đồng thời. Do đó cần phải có máy tính chứa nhiều bộ xử lý. Khi đó mỗi bộ xử lý sẽ được phân công thực hiện các công đoạn này. Những máy song song MPP (Machine Massively Parallel Processors) được ra đời do những yêu cầu này. Máy tính MPP tổ hợp hàng trăm cho đến hàng ngàn bộ xử lý (CPU) với bộ nhớ lên tới hàng trăm Gb trên đường truyền (BUS) với tốc độ rất cao. Do đó được dùng để giải quyết các bài toán lớn trên thực tế như "Bài toán dự báo thời tiết", "Thiết kế và mô phỏng hoạt động của các vi mạch trong thời gian thực",…

Một số mô hình kiến trúc của máy song song MPP là mô hình mảng tuyến tính 1 chiều, mảng vòng 1 chiều, mảng tuyến tính 2 chiều, mảng vòng 2 chiều, 1D Hybercube, 2D Hybercube, 3D Hybercube, 4D Hybercube,... Các mô hình này được phân loại dựa vào topology của liên kết vật lý bên trong giữa các bộ vi xử lý.Vì liên kết bên trong là cố định nên kiến trúc của các máy song song MPP là không đổi. Trong khi các bài toán trong thực tế có mô hình xử lý song song của các công đoạn thường phong phú đôi khi không phù hợp với kiến trúc hiện có của máy tính MPP, dẫn đến không tận dụng hết được khả năng của máy MPP.

2.1.2. Máy trạm thay thế (Cluster of Workstation)

Để khắc phục được nhược điểm của máy MPP, người ta dùng máy trạm thay thế. Các máy trạm thông thường có kiến trúc đơn giản gồm 1 hoặc vài bộ xử lý và bộ nhớ cục bộ khoảng vài chục Mb được liên kết lại thành một mạng máy tính. Cũng giống như máy tính song song MPP, các công đoạn của bài toán sẽ được phân công xử lý trên các máy trạm này. Dữ liệu vào/ ra sẽ được truyền trên mạng. Tuy nhiên tốc độ truyền dữ liệu trên mạng không nhanh bằng tốc độ của liên kết vật lý bên trong của MPP nhưng nó có ưu điểm là có kiến trúc uyển chuyển hơn và giá thành rẻ hơn.

2.1.3. Tính toán trên mạng không đồng nhất

Trong MPP, tất cả các bộ xử lý đều giống nhau về tốc độ, dung lượng bộ nhớ, tốc độ truyền thông. Ngược lại, trong mạng máy tính thì hoàn toàn khác. Các máy tính trạm có thể khác nhau về kiến trúc, tốc độ, dung lượng bộ nhớ,..Vì thế một chương trình muốn khai thác bộ máy tính mạng này phải giải quyết các vấn đề khác nhau giữa:

- Kiến trúc.
- Khuôn dạng dữ liệu (khi chuyển đổi dữ liệu qua lại giữa hai trạm).
- Tốc độ tính toán.
- Tải của các máy (Machine workload).
- Tải của mạng (Network workload).

Tuy nhiên việc tính toán trên mạng không đồng nhất cũng có lợi thế:

Tận dụng các máy trạm nhàn rỗi.

- Hiệu suất có thể cải thiện nhờ tối ưu hóa việc phân công công đoạn cho các máy trạm phù hợp nhất.
- Khai thác được bản chất không đồng nhất của tính toán. Một mạng tính toán không chỉ gồm các máy trạm kết nối với nhau trên một mạng cục bộ mà có thể tận dụng khả năng của các máy trên mạng diện rộng (Internet).
- Tốc độ mạng ngày càng được cải thiện làm khả năng hiện thực và số lượng bài toán giải bằng mô hình này ngày càng tăng.

2.2. Kiến trúc của máy ảo song song PVM (Parallel Virtual Machine)

2.2.1. Định nghĩa

PVM là một bộ phần mềm tích hợp nhằm mô phỏng một mô hình tính toán phân tán mềm dẻo và đa năng trên mạng máy tính không đồng nhất.

2.2.2. Nguyên lý của một hệ thống PVM

Một hệ thống PVM dựa trên tập máy chủ cấu hình bởi người sử dụng. Các công đoạn tính toán của ứng dụng sẽ được xử lý phân tán trên những máy chủ này, thậm chí có thể thêm bớt trong khi chạy chương trình. Các máy chủ này có thể là một máy trạm độc lập hay một máy tính MPP.

Khi một máy chủ (host) gia nhập vào máy ảo song song thì các tài nguyên của máy này sẽ trở thành tài nguyên của hệ thống. Vì vậy các công việc (task) khi được phân công xử lý trên host này đều có thể tận dụng hết tài nguyên trên máy đó thậm chí cả hệ thống tập tin.

Một số đặc điểm của máy ảo PVM:

- *Dựa trên tiến trình:* Đơn vị tính toán song song trong PVM là task. Task là một đoạn mã tuần tự độc lập và sẽ được ánh xạ tới các host khi đang chạy chương trình ứng dụng.
- *Mô hình truyền thông điệp (message – passing model):* Dữ liệu trao đổi giữa các task trong hệ thống dựa vào cơ chế truyền thông điệp. Kích thước của thông điệp chỉ bị giới hạn do tài nguyên hệ thống.
- *Hỗ trợ trên mạng không đồng nhất:* PVM cho phép thông điệp chứa nhiều kiểu dữ liệu để trao đổi giữa các host có các dạng biểu diễn dữ liệu khác nhau.
- *Hỗ trợ máy tính MPP:* PVM vẫn sử dụng cơ chế truyền thông điệp cũ trên các máy MPP gồm nhiều bộ xử lý để tận dụng lợi thế của phần cứng.
- *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 người sử dụng.

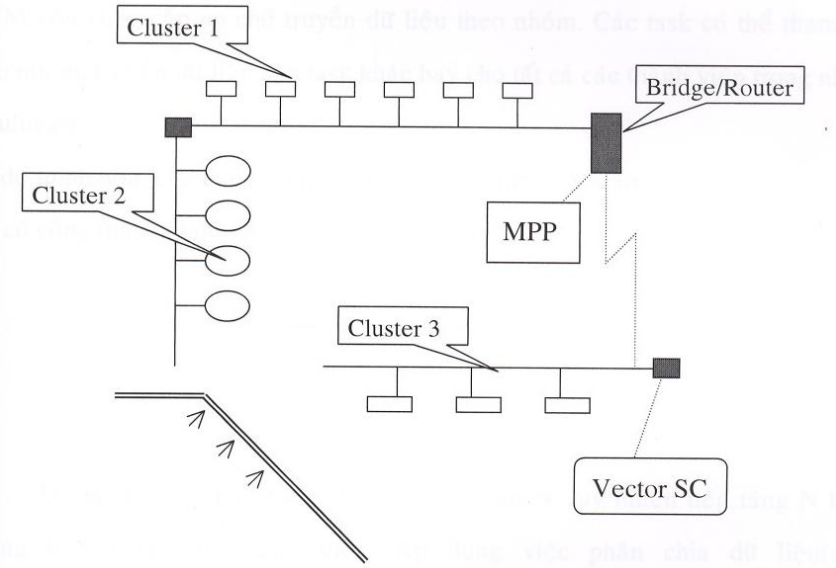
2.2.3. Cấu trúc của PVM

Hệ thống của PVM gồm 2 phần:

- *Phần hạt nhân (pvmd/pvm3):* Là một tiến trình thường trú (deamon) được đặt trên tất cả các máy tính để tạo ra máy ảo. Nó được thiết kế để bất cứ người dùng đăng nhập hợp lệ có thể cài đặt tiến trình này trên máy tính. Khi người dùng muốn chạy một ứng dụng PVM, việc đầu tiên là phải tạo ra máy ảo bằng cách bắt đầu một PVM. Các ứng dụng PVM sau đó có thể được bắt đầu từ một dấu nhắc trên Unix ở bất kỳ một máy chủ nào. Nhiều người dùng có thể cấu hình lên nhiều máy ảo và mỗi người dùng có thể thực thi các ứng dụng PVM cùng một lúc. Phần này sẽ được khởi tạo cùng với máy ảo song song PVM trên tất cả các host. Nó chịu trách nhiệm quản lý các task PVM trên từng host: đảm bảo quá trình truyền thông điệp tới các task trên máy chủ.

- *Phần thư viện các chương trình con giao diện lập trình của PVM:* gồm các chương trình con trong bộ thư viện PVM. Thư viện này bao gồm các thủ tục truyền thông điệp, sinh các task, điều phối các task và thay đổi cấu hình của máy ảo PVM.

2.2.4. Kiến trúc của PVM



Hình 2.1. Kiến trúc của PVM

Hình trên mô tả kiến trúc điển hình của máy ảo PVM trên mạng không đồng nhất. Các máy trạm trên cùng một trạm cục bộ (LAN) được nhóm thành một Cluster. Trong một Cluster có thể chứa nhiều máy trạm có cấu hình khác nhau. Thậm chí một máy tính song song MPP cũng có thể trở thành thành viên của một Cluster. Các Cluster được liên kết với nhau thông qua cầu nối Bridge/ Router.

2.3. Cơ chế hoạt động

Một ứng dụng chạy trên PVM được phân chia thành nhiều task. Mỗi task thực hiện một phần công việc. Trong mỗi task sẽ chứa các thủ tục của PVM để sinh ra các task khác, truyền dữ liệu với các task khác hay đồng bộ hóa với nhiều task khi ứng dụng được thực thi. Mỗi task khi sinh ra được gán với một số hiệu duy nhất gọi là TaskID. Mọi dữ liệu truyền giữa 2 task thông qua số hiệu này. Mô hình PVM luôn đảm bảo thứ tự truyền của thông điệp giữa 2 task bất kỳ. Ngoài ra PVM còn cung cấp cơ chế truyền dữ liệu theo nhóm. Các task có thể tham gia vào nhóm, truyền dữ liệu cho task khác hay cho tất cả các thành viên trong nhóm (Multicast).

Phương thức thực hiện chương trình trong PVM như sau:

- 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 đều truy cập được.
- Người sử dụng tạo ra một bản sao của tác vụ chủ (master) hoặc khởi động một tác vụ. Một tiến trình được khởi động bởi một tiến trình khác được gọi là tiến trình tớ (slave). Những tiến trình này thực hiện một số tính toán cục bộ và trao đổi với nhau để giải quyết bài toán đặt ra.

Để cài đặt một thuật toán tính toán thông thường, sử dụng mô hình Master – Slave ta có một số bước cơ bản như sau:

Chương trình Master sẽ làm các nhiệm vụ:

- Sinh ra NHOSTS chương trình tính toán (Slave) trên mỗi máy trạm (NHOSTS là số máy trạm hiện có).
- Gửi các dữ liệu tính toán, ví dụ như số bước lặp N, số thứ tự của mỗi chương trình tính toán (nproc),…
- Nhận kết quả sau khi các chương trình tính toán xong.
- Tính tổng và hiển thị kết quả.

Mỗi chương trình tính toán Slave sẽ làm các nhiệm vụ:

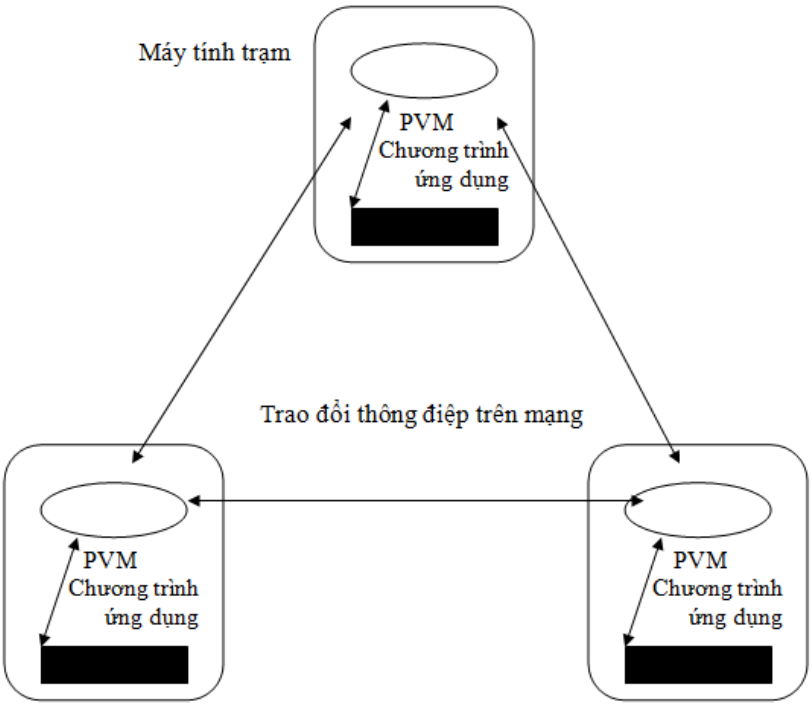
- Nhận dữ liệu từ chương trình Master.
- Thực hiện các phép tính.
- Gửi trả kết quả tính được cho chương trình Master.

2.4. Lập trình trên cụm máy tính PVM

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

Tập các máy tính được sử dụng trong mạng phải được định nghĩa theo các mức ưu tiên để chạy các chương trình. Điều này được thực hiện trên tập máy ảo song song PVM. 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 thông dữ liệu.



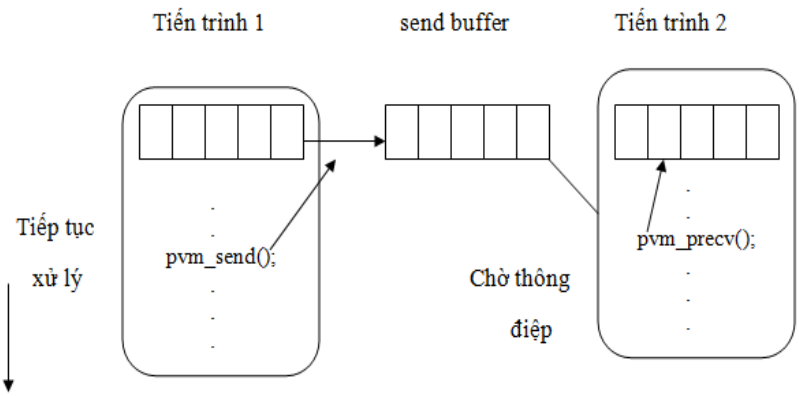
Hình 2.2. Sự trao đổi thông điệp của các máy tính trong hệ PVM

Các chương trình của PVM thường được tổ chức theo mô hình chủ - tớ (master - slave), trong đó tiến trình chủ được thực hiện trước tiên, sau đó các tiến trình tớ sẽ được tạo ra trong tiến trình chủ đó. Hàm phát sinh tiến trình mới trong PVM là: pvm_spawn().

Một tiến trình muốn tham gia vào hệ PVM thì nó phải ghi danh bằng cách gọi hàm pvm_mytid(). Các tiến trình muốn được huỷ bỏ thì gọi hàm pvm_exit().

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

Nếu dữ liệu được gửi đi là một danh sách các mục có cùng kiểu thì trong PVM sử dụng pvm_psend() và pvm_precv().Để mô tả hoạt động của hai tiến trình trao đổi một mảng dữ liệu với nhau ta có hình 2.3.



Hình 2.3. Gọi hàm pvm_psend() và pvm_precv()

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

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

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

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

Để dễ hiểu,ta có thể tóm tắt lại giao diện lập trình PVM như sau:

2.4.1. Điều khiển các task

- pvm_mytid(): Trả lại số hiệu của task hiện hành.
- pvm_exit(): Thoát ra khỏi môi trường của máy ảo song song.
- pvm_spawn(…): Sinh ra một task con PVM.
- pvm_parent(): Trả lại số hiệu của task cha (task đã sinh ra task hiện hành).

2.4.2. Truyền thông điệp

Các thông điệp truyền được lưu trong các bộ đệm. Có 3 bước để truyền một thông điệp:

- Khởi tạo thông điệp bằng thủ tục pvm_initsend(…).

- Dữ liệu sẽ được đặt vào thông điệp bằng các thủ tục pvm_pk*() tùy theo kiểu dữ liệu.
- Gửi thông điệp bằng pvm_send(…) cho một task cụ thể hay pvm_mcast cho nhiều task cùng một lúc.

2.4.3. Nhận thông điệp

Các bước để nhận một thông điệp từ task khác:

- Kiểm tra thông điệp tới:
  - pvm_recv(…): Sẽ đợi cho đến khi thông điệp từ task mong muốn gửi tới.
  - pvm_nrecv(…): Kiểm tra xem thông điệp từ task mong muốn đã gửi tới chưa.
  - pvm_trecv(…): Đợi thông điệp gửi tới trong một khoảng thời gian đã cho nếu lâu quá thì sẽ tiếp tục thực thi các lệnh tiếp theo.
- Lấy dữ liệu từ thông điệp pvm_upk*() tùy theo kiểu dữ liệu được đưa vào khi gửi.

2.4.4. Nhóm các task

- Các task có thể gộp thành nhóm bằng:

  pvm_joingroup (tên nhóm);

- Ra khỏi nhóm bằng:

  pvm_lvgroup (tên nhóm);

Một task có thể cùng một lúc gia nhập vào nhiều nhóm. Khi task đang ở trong nhóm có thể lấy các thông tin về nhóm như: kích thước hiện hành của nhóm, danh sách các task trong nhóm.

- pvm_gettid(…): Trả về số hiệu task trong nhóm.
- pvm_getinst(…): Trả về thứ tự của task trong nhóm.
- pvm_gsize(…): Kích thước nhóm: số các task hiện có trong nhóm.
- pvm_barrier(…): Đồng bộ các task trong nhóm.
- pvm_bcast(…): Truyền thông điệp cho tất cả các thành viên trong nhóm.

2.4.5. Các kiểu dữ liệu được đóng gói trong PVM

Khi dữ liệu trong thông điệp được truyền từ platform này sang platform khác, PVM sẽ tự động chuyển đổi các khuôn dạng kiểu dữ liệu. PVM hỗ trợ các kiểu dữ liệu cơ bản sau:

- Kiểu số nguyên có dấu và không dấu:
  - Char: 1bytes.
  - Short: 2bytes.
  - Int: số nguyên 4 bytes.
  - Long: 8bytes
- Kiểu số thực có dấu và không dấu:
  - Float
  - Double
- Kiểu dữ liệu khác:
  - Char*: chuỗi ký tự.
  - Cpl*: số phức (double, double).

2.5. Sử dụng PVM

PVM cung cấp các thủ tục để khởi tạo các tác vụ trên máy ảo (virtual machine) và cho phép các tác vụ này trao đổi với nhau.

2.5.1. Cài đặt PVM

Phiên bản mới nhất của mã nguồn PVM và tài liệu luôn luôn có sẵn thông qua netlib. Netlib là một dịch vụ phân phối phần mềm thành lập trên Internet có chứa một loạt các phần mềm máy tính. Phần mềm có thể được lấy từ netlib bằng ftp, WWW, xnetlib, hoặc email. Download pvm3.4.6 có trên Internet tại địa chỉ http://www.netlib.org/pvm3/pvm3.4.6.tgz.

Thông thường PVM được cài đặt để nhiều người cùng sử dụng, hoặc cho nhiều đề án khác nhau của cùng một người. Trong cả hai trường hợp PVM đều có mục tiêu sử dụng chung.

PVM sử dụng hai biến môi trường khi bắt đầu và chạy chương trình. Mỗi người dùng PVM cần thiết lập hai biến này để có thể sử dụng.

- *Biến đầu tiên là PVM_ROOT:* Biến này thiết lập vị trí của thư mục cài đặt pvm3.
- *Biến thứ hai là PVM_ARCH:* Khai báo kiến trúc của máy chủ PVM và chọn một số thực thi từ thư mục PVM_ROOT.

Phương pháp đơn giản nhất là thiết lập hai biến này trong tập tin .cshrc. Giả sử chúng ta sử dụng csh theo phương pháp trên, ta có ví dụ cho việc thiết lập PVM_ROOT : setenv PVM_ROOT $ HOME/pvm3.

Cần đặt đường dẫn và các biến môi trường sau đây trong tập tin /etc/profile (để sử dụng chung) hoặc trong $HOME/.bashrc.

Để PVM daemon hoạt động được trên nhiều node tạo nên máy ảo PVM trong tập tin /etc/hosts.equiv, hoặc trong các tập tin $HOME/.rhosts.

Biến môi trường cũng phải được đặt trong user tương ứng, chẳng hạn trong tập tin $HOME/.bashrc.

Các chương trình thi hành bằng lệnh pvm_spawn() phải được chỉ đường dẫn tuyệt đối, hoặc được lưu trữ trong thư mục $PVM_ROOT/bin/$PVM_ARCH.

Tóm lại, để cài đặt PVM ta có các bước sau:

- Đặt PVM_ROOT và PVM_ARCH trong tập tin cshrc.
- Xây dựng PVM đối với từng loại kiến trúc.
- Tạo một tập tin .rhosts trên mỗi máy chủ để liệt kê tất cả các host muốn sử dụng.
- Tạo một tập tin $HOME/.xpvm_hosts để liệt kê tất cả các host.

Nguồn PVM đi kèm với các thư mục và makefiles cho hầu hết các kiến trúc. Xây dựng đối với từng loại kiến trúc được thực hiện tự động bằng cách đăng nhập vào một máy chủ, đi vào thư mục PVM_ROOT, và đánh máy làm. Các makefile sẽ tự động xác định những kiến trúc nó đang được thực thi trên, tạo ra các thư mục con thích hợp, và xây dựng PVM, pvmd3, libpvm3.a, và libfpvm3.a, pvmgs, và libgpvm3.a. Nó đặt tất cả những tập tin này trong $PVM_ROOT/lib/PVM_ARCH, ngoại trừ pvmgs được đặt trong $PVM_ROOT/bin/ PVM_ARCH.

2.5.2. Bắt đầu với PVM

Trên bất kỳ máy chủ mà trên đó PVM đã được cài đặt, ta có thể gõ : %pvm để bắt đầu.

Khi đó sẽ nhận lại một dấu nhắc có nghĩa là PVM hiện đang chạy trên máy chủ này. Ta có thể làm một số thao tác sau:

- PVM > add hostname : Thêm máy chủ cho máy ảo.
- PVM > delete hostname: Xóa các host từ máy ảo.

Nếu nhận được thông báo "Can't start pvmd" thì kiểm tra vấn đề khởi động và thử lại.

- PVM > conf: Kiểm tra cấu hình máy ảo hiện tại.
- PVM > ps – a: Xem những tác vụ đang chạy trên máy ảo.
- PVM > quit: Đóng giao diện điều khiển PVM nhưng máy ảo và những công việc vẫn tiếp tục chạy. Và muốn tiếp tục với máy ảo, ta có thể gõ %PVM.
- PVM > halt: Kết thúc với các máy ảo. Lệnh này kết thúc bất cứ tác vụ nào đang chạy của máy ảo PVM, tắt máy ảo và thoát khỏi giao diện điều khiển. Đây là phương pháp để dừng máy ảo, nó đảm bảo rằng các máy ảo được tắt hoàn toàn.
- Nếu không muốn gõ một loạt các tên máy chủ mỗi lần, chỉ việc nhập : %PVM hostfile. PVM sẽ thêm tất cả các máy chủ được liệt kê cùng một lúc trước khi dấu nhắc xuất hiện trên giao diện điều khiển. Một số tùy chọn có thể được xác định trên cơ sở mỗi máy chủ trong hostfile. Ngoài ra người dùng có thể tùy chỉnh máy ảo của mình cho một ứng dụng hoặc một môi trường cụ thể.

2.5.3. Một số vấn đề khi sử dụng PVM

- Nếu có thông báo là: [t80040000] Can't start pvmd thì việc đầu tiên là phải kiểm tra file .rhosts trên máy chủ từ xa chứa tên máy chủ PVM đang làm việc. Ngoài ra kiểm tra xem tập tin .rhosts đã được thiết lập một cách chính xác chưa bằng cách nhập cú pháp:

```
% rsh remote_host ls
```

Nếu .rhost được thiết lập một cách chính xác thì sẽ thấy một danh sách các tập tin trên máy chủ từ xa.

Một lý do khác là không có PVM được cài đặt trên máy chủ hoặc không có PVM_ROOT được thiết lập chính xác ở một số máy chủ. Ta có thể kiểm tra bằng cách gõ:

```
%  rsh remote_host $PVM_ROOT/lib/pvmd
```

- Nếu thông báo Login Incorrect, điều này có nghĩa là không có tài khoản trên máy chủ từ xa với tên đăng nhập đó. Nếu sử dụng một tên đăng nhập khác trên máy chủ từ xa thì phải sử dụng tùy chọn "io=" trong hostfile.
- Nếu nhận được bất cứ một thông báo lạ nào thì kiểm tra tập tin .cshrs.

2.5.4. Chạy chương trình PVM

Phần này sẽ tìm hiểu cách biên dịch và chạy chương trình PVM.

Sao chép một chương trình ví dụ vào thư mục riêng cần lưu trữ:

```
% cp -r $PVM_ROOT/examples $HOME/pvm3/examples
% cd $HOME/pvm3/examples
```

Các thư mục examples có chứa một file Readme là makefile.aimk và nó mô tả cách xây dựng những ví dụ này. Aimk tự động được thêm vào $PATH khi đặt cshrc.stub vào trong tập tin .cshrc. Sử dụng aimk cho phép để lại mã nguồn và makefile không thay đổi khi biên dịch trên những kiểu kiến trúc khác nhau.

Mô hình lập trình Master/ Slave (Chủ/ Tớ) là mô hình phổ biến nhất được sử dụng trong tính toán phân tán. (Trong lĩnh vực lập trình song song nói chung, mô hình SPMD là phổ biến hơn). Để biên dịch bằng ngôn ngữ C, ta làm như sau:

```
%aimk master slave
```

Makefile di chuyển các file thực thi đến $HOME/pvm3/bin/PVM_ARCH, đây là vị trí mặc định PVM sẽ tìm chúng trên tất cả các host. Nếu file hệ thống không giống nhau trên tất cả các host PVM thì sẽ phải xây dựng hoặc sao chép các file thực thi (tùy thuộc vào từng kiến trúc) trên tất cả các host PVM.

Ví dụ: Từ một cửa sổ, mở PVM để bắt đầu và cấu hình các host. Trong cửa sổ khác, cd đến $HOME/pvm3/bin/PVM_ARCH và gõ:

```
%master
```

Khi đó chương trình sẽ hiện ra số tác vụ, số lượng tác vụ không có để phù hợp với số lượng máy chủ,..

Điều này cho thấy khả năng chạy một chương trình PVM từ một dấu nhắc dòng lệnh trên bất kỳ máy chủ trong máy ảo nào.

2.5.5. Giao diện điều khiển PVM

Giao diện điều khiển PVM (được gọi là pvm) là một nhiệm vụ độc lập cho phép người dùng khởi động, truy vấn và sửa đổi các máy ảo. Giao diện điều khiển có thể được bắt đầu và dừng lại nhiều lần trên bất kỳ máy chủ trong các máy ảo mà không ảnh hưởng đến PVM hoặc bất kỳ ứng dụng nào đang chạy.

Khi được khởi động, pvm xác định PVM đã sẵn sàng chạy chưa. Nếu chưa sẵn sàng chạy, pvm sẽ tự động thực thi pvmd trên máy chủ này, bỏ qua những tùy chọn dòng lệnh pvmd và hostfile. Như vậy, không cần thiết phải chạy PVM để bắt đầu giao diện điều khiển.

pvm [-n <hostname>][hostfile]

Khi đó, tùy chọn –n là chỉ định một tên khác cho cụm máy chủ pvmd (trong trường hợp tên máy không phù hợp với địa chỉ IP). Khi PVM được bắt đầu, giao diện điều khiển sẽ hiện ra dấu nhắc:

pvm>

và chấp nhận các lệnh từ chuẩn vào. Một số các lệnh chuẩn:

- add: Thêm máy chủ cho máy ảo, theo sau là một hoặc nhiều host.
- alias: định nghĩa hoặc liệt kê các lệnh.
- conf: Liệt kê các cấu hình của máy ảo bao gồm tên máy, tác vụ pvmd, loại kiến trúc, tốc độ đánh giá,…
- delete: Xóa các máy chủ từ máy ảo, theo sau là một hoặc nhiều host. PVM vẫn được chạy trên các host bị mất.
- echo: Đối số echo.
- halt: Kết thúc tất cả các xử lý của PVM, bao gồm giao diện điều khiển và sau đó tắt PVM. Tất cả các tiến trình thường trú đều được tắt bởi lệnh này.
- help: Được sử dụng để lấy thông tin về các lệnh tương tác. Trợ giúp này có thể được theo sau bởi một tên lệnh liệt kê các tùy chọn.
- id: In các tác vụ của giao diện điều khiển.
- jobs: Danh sách công việc đang chạy.
- kill: Được sử dụng để chấm dứt bất kỳ một xử lý nào của PVM.
- mstat: Cho biết trạng thái của các host cụ thể.

- quit: Thoát khỏi giao diện điều khiển, rời khỏi tiến trình thường trú và các công việc đang chạy.
- reset: Hủy tất cả các xử lý của PVM trừ giao diện điều khiển và thiết lập lại PVM nội bộ và hàng đợi tin nhắn.
- sentenv: hiển thị hoặc thiết lập các biến môi trường.
- sig: Gửi các tín hiệu tới tác vụ, theo sau là tín hiệu số và TID.
- spawn: Bắt đầu một ứng dụng PVM. Theo sau là các tùy chọn:
  - count: Số các tác vụ, mặc định là 1.
  - host
  - ARCH: Sinh ra các máy chủ loại ARCH.
  - ?: Cho phép gỡ rối.
  - >: Chuyển hướng đầu ra tác vụ cho giao diện điều khiển.
  - >file: Chuyển hướng đầu ra tác vụ tới tệp tin.
  - >>file: Chuyển hướng đầu ra tác vụ nối thêm tới tệp tin.
  - @: Theo dõi công việc, hiển thị đầu ra trên giao diện điều khiển.
  - @file: Theo dõi công việc, hiển thị đầu ra trên tệp tin.
- trace: Thiết lập và hiển thị các sự kiện.
- unalias: Hủy lệnh bí danh.
- version: In phiên bản PVM đang được sử dụng.

PVM hỗ trợ sử dụng nhiều giao diện điều khiển. Có thể chạy một giao diện điều khiển trên bất kỳ máy chủ nào trong một máy ảo hiện có và thậm chí nhiều giao diện điều khiển trên một máy. Cũng có thể khởi động giao diện điều khiển ở giữa một ứng dụng PVM và kiểm tra tiến độ của nó.

2.5.6. Các tùy chọn của hostfile

Như đã biết, chỉ cần một người phải cài đặt PVM, nhưng mỗi người dùng PVM có thể có hostfile của riêng mình. Các hostfile xác định cấu hình của các máy chủ mà PVM sẽ kết hợp lại thành một máy ảo. Nó cũng chứa thông tin về những máy chủ mà người dùng có thể muốn thêm vào để cấu hình.

Các hostfile đơn giản chỉ là một danh sách các tên máy chủ trên một dòng lệnh. Dòng trống sẽ được bỏ qua, những dòng bắt đầu với "#" là những dòng bình luận.

Có một số tùy chọn sau: (Các tùy chọn được phân cách bằng khoảng trống)

- lo = userid: Cho phép người dùng chỉ định một tên đăng nhập thay thế cho host này. Nếu không thì máy tính sẽ sử dụng tên đăng nhập cũ.
- so = pw: Nhắc nhở người dùng cho mật khẩu trên host này. Điều này rất hữu ích trong trường hợp người dùng có một userid và mật khẩu khác nhau trên hệ thống từ xa. PVM sử dụng rsh mặc định để khởi động từ xa pvmd. Khi pw được xác định, PVM sẽ sử dụng rexec() để thay thế.
- dx = location of pvmd: Cho phép người dùng chỉ định một vị trí khác với vị trí mặc định trên host này. Điều này rất hữu ích trong trường hợp người dùng muốn sử dụng bản sao pvmd của chính mình.
- ep = đường dẫn đến file thực thi của người dùng: Cho phép người dùng chỉ định một loạt các đường dẫn để tìm các tập tin yêu cầu. Nhiều path được phân cách bằng dấu hai chấm. Nếu không chỉ định ep này thì PVM sẽ tìm kiếm trong $HOME/pvm3/bin/PVM_ARCH cho các tác vụ ứng dụng.
- sp = value: Xác định tốc độ tính toán tương đối của máy chủ này với máy chủ khác trong một cấu hình. Phạm vi giá trị có thể là từ 1 đến 1000000, mặc định là 1000.
- bx = location of debugger: Chương trình gỡ rối mặc định là pvm3/lib/debugger.
- wd = working_directory: Chỉ định một thư mục làm việc mà trong đó tất cả các nhiệm vụ sinh ra trên host này sẽ được thực hiện. Mặc định là $HOME.
- ip = hostname: Xác định một tên khác để giải quyết đến địa chỉ IP của máy chủ.
- so = ms: Xác định rằng pvmd ở máy slave sẽ được bắt đầu một cách thông thường trên host này. Điều này rất hữu ích khi rsh và dịch vụ mạng rexec bị vô hiệu hóa nhưng vẫn tồn tại kết nối IP. Khi sử dụng tùy chọn này, người dùng sẽ thấy trong tty của pvm3.

Nếu người dùng muốn thiết lập các tùy chọn trên là mặc định cho một loạt các host thì có thể đặt các tùy chọn này trên một dòng duy nhất với " * " ở đầu dòng

cho trường tên máy. Các mặc định sẽ có hiệu lực cho tất cả các host sau cho đến khi chúng được ghi đè bởi một dòng thiết lập mặc định khác.

Những máy chủ chưa được cấu hình ban đầu có thể cấu hình lại được trong hostfile bởi những dòng lệnh bắt đầu từ " & ".

2.6. Lập trình dùng PVM

- Để làm việc, chương trình thi hành phải được pvmd khởi động trên các node của máy ảo.
- Trên mỗi node có thể có nhiều chương trình thực thi cùng hoạt động.

2.6.1. Mô hình Master – Slave

- Khi dùng mô hình master – slave, trong PVM viết hai chương trình riêng.
- Chương trình thứ nhất đóng vai trò master bao gồm các phần như sau:
  - Mô tả prototype các hàm PVM để sử dụng

    #include "pvm3.h"
  - Kích hoạt chương trình slave bởi pvm_spawn()
  - Gửi dữ liệu cho các tiến trình slave bởi lần lượt các hàm

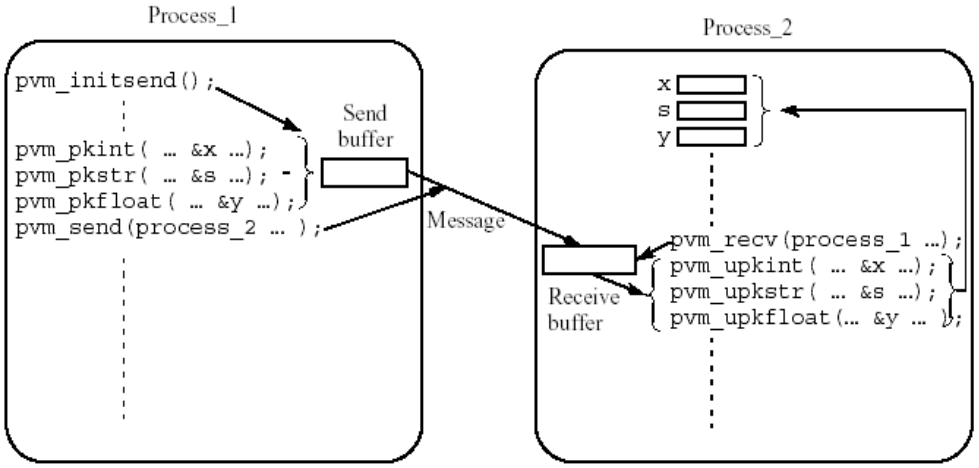
    pvm_initsend()

    pvm_packXXX()

    pvm_send()
  - Nhận dữ liệu từ các tiến trình slave bởi lần lượt các hàm

    pvm_recv()

    pvm_upackXXX()
  - Thoát tiến trình khỏi PVM bởi hàm pvm_exit()
- Chương trình slave bao gồm:
  - Mô tả prototype để sử dụng #include "pvm3.h"
  - Xác định ID của tiến trình master: pvm_parent().
  - Truyền dữ liệu cho tiến trình master.
  - Thoát tiến trình khỏi PVM bởi hàm pvm_exit()



Hình 2.4. Phương thức trao đổi các tiến trình

2.6.2. Mô hình Task – to – Task

- Trong trường hợp này chỉ sử dụng 1 chương trình, trong chương trình bao gồm các phần như:
  - Mô tả prototype các hàm PVM để sử dụng

    #include "pvm3.h"
  - Xác định ID của tiến trình đang hoạt động bởi hàm pvm_mytid() và pvm_parent().
  - Nếu chưa có thực thể nào thi hành, hàm pvm_parent() trả về giá trị là PvmNoParent. Với giá trị này, dùng để điều khiển đoạn chương trình của master. Trong đoạn này cũng có các hàm chuyển thông điệp.
  - Trong phần không phải master, sử dụng các hàm truyền dữ liệu thích hợp với phần master.
  - Thoát tiến trình khỏi PVM bởi hàm pvm_exit()
- int pvm_mytid(void):
  - Trả về tid (task Id) của tiến trình đang gọi hàm này,
  - Nếu pvm chưa kích hoạt giá trị trả về của hàm là số âm.
- int pvm_parent(void):
  - Trả về tid của tiến trình kích hoạt tiến trình đang gọi hàm này.
  - Nếu thực thể đang gọi hàm này không kích hoạt hàm pvm_spawn(), giá trị trả về là PvmNoParent. Nếu giá trị là PvmSysErr cho chúng ta biết không liên kết được với pvmd trên node địa phương này.

- int numt = pvm_spawn(char* task, char** args, int flag, char* where, int ntasks, int* tids):
  - Khởi động các tiến trình PVM mới.
  - task: chuỗi ký tự chỉ tên tập tin cần kích hoạt, tập tin này đặt ở đường dẫn chỉ định của PVM (chuẩn trong trường hợp cài đặt này là $PVM_ROOT/bin/$PVM_ARCH) hoặc đường dẫn tuyệt đối.
  - args: mảng các chuỗi ký tự, là các tham số trên dòng lệnh của tập tin thi hành.
  - flag: biến nguyên, có thể nhận các giá trị như:
    - ✓ PvmTaskDefault: Có thể khởi động bất kỳ máy nào
    - ✓ PvmTaskHost: Chỉ định máy để khởi động
    - ✓ PvmTaskArch: Chỉ định loại kiến trúc máy để khởi động.
  - where: mảng ký tự chỉ định tên máy khi flag có giá trị là PvmTaskHost, nếu không giá trị là NULL.
  - ntasks: số tiến trình cần kích hoạt
  - tids: biến con trỏ lưu trữ các tid của những tiến trình đã kích hoạt.
  - numt: nếu bằng ntasks sự kích hoạt thành công, nếu bằng giá trị âm hoặc nhỏ hơn ntasks sự kích hoạt còn gặp sai sót.

Ví dụ:

```
char* args[] = { "4", "100", (char*)0};
int numt = pvm_spawn( "slave1", args, PvmTaskHost, "p1.ioit.ac.vn", 10, &tids );
```

- int pvm_initsend(int encode):
  - Khởi động quá trình truyền, với tham số encode chỉ định sơ đồ mã hoá thông điệp truyền đi.
  - Các giá trị của encode có thể là PvmDataDefault với kiểu mã hoá XDR; PvmDataRow không mã hoá, v.v...
- int pvm_send(int tid, int tag): Để gửi dữ liệu có trong vùng đệm thông điệp đến tiến trình tid với nhãn tag. Đây là hàm gửi không đồng bộ.

- int pvm_recv(int tid, int tag): Đây là hàm nhận đồng bộ, giá trị trong vùng đệm thông điệp được gử đến tid với nhãn tag chỉ định.
- int pvm_pack(): có các hàm tương ứng như sau:
  - pvm_pkint(int *buff, int nitems, int stride): để chuẩn bị gửi nitems dữ liệu tại địa chỉ trong buff, với bước sải là stride (chẳng hạn gửi các phần tử ở vị trí chẵn của buff thì stride là 2).
  - pvm_pkfloat(float *buff, int nitems, int stride): tương tự
  - pvm_pkdouble(double *buff, int nitems, int stride): tương tự
  - pvm_pkstr(char* buff): chuẩn bị gửi chuỗi ký tự buff.
- int pvm_unpack(): gồm các hàm
  - pvm_upkint(),
  - upkfloat(), ...

với các tham số giống hàm pvm_pack().

- int pvm_exit(void): thoát tiến trình ra khỏi PVM.
- int pvm_bcast(char *group, int tag): Phát tán dữ liệu thông điệp đang tác động đến nhóm các tiến trình trong group.
- Để dùng trước đó phải tạo group bởi pvm_jointgroup(char * group).
- int pvm_mcast(int tids, int ntasks, int tag): Phát tán dữ liệu đến tập hợp các tiến trình chỉ định.
- int pvm_reduce (void (*func)(), void *data, int count, int datatype, int tag, char *group, int root):
  - data : mảng chứa dữ liệu trên tiến trình.
  - count : số tiến trình.
  - datatype : kiểu dữ liệu có thể là PVM_INT, PVM_FLOAT, ...
  - root : nơi tập hợp.
  - func : hàm dùng tập hợp, có các hàm đã định nghĩa như PvmSum, PvmProduct, PvmMax, PvmMin.
  - Đây là hàm nhận không đồng bộ.

- int pvm_scatter(void *result, void *data, int count, int datatype, int tag, char *group, int root): Để phân phát dữ liệu data trên root đến các tiến trình trong nhóm group.
- int pvm_gather(void *result, void *data, int count, int datatype, int tag, char *group, int root): Để thu thập dữ liệu từ các tiến trình trong group về mảng result trong root. Đây là hàm nhận không đồng bộ.

2.7. Thiết kế môi trường hỗ trợ tính toán song song

Để thiết kế một môi trường hỗ trợ cho lập trình song song ta phải xem xét các yếu tố gây khó dễ cho người lập trình song song. Đó là:

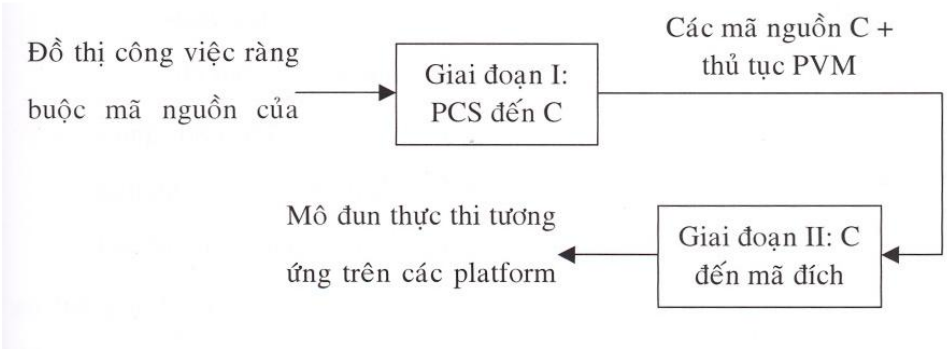
- Biểu diễn thuật toán song song.
- Phân chia công việc.
- Quản lý việc khởi tạo (sinh) và kết thúc các công việc.
- Đồng bộ hóa giữa các công việc.
- Quản lý việc trao đổi thông tin giữa các task.
- Quản lý mã nguồn chương trình, mã thực thi trên môi trường xử lý phân tán trên nhiều flatform khác nhau.
- Quản lý biến phân chia.
- Tối ưu việc phân công công việc cho các bộ xử lý.
- Xử lý, gỡ rối chương trình.
- Đánh giá thời gian thực thi, giúp người lập trình hiệu chỉnh lại thuật toán.
- Phát hiện những hư hỏng và khôi phục, sửa chữa như connection lost, host failure,…trong máy ảo song song PVM.

Một môi trường hỗ trợ tốt là khắc phục càng nhiều khó khăn cho người lập trình càng tốt.

Để biểu diễn thuật toán song song, nên sử dụng công cụ biểu diễn trực quan bằng đồ thị công việc với mỗi nút là một module chức năng tính toán, các cung mô tả các ràng buộc như dữ liệu vào / ra, luật kích hoạt các công việc tiếp theo, qua đó đồng bộ hóa giữa các công việc.

Sau khi phân tích thuật toán song song và biểu diễn trên đồ thị công việc, người lập trình cần được cung cấp các công cụ lập trình như bộ soạn thảo mã nguồn, bộ biên dịch, mô tả các dữ liệu trao đổi giữa các task,…

Bộ biên dịch thực hiện nhiệm vụ biên dịch lại toàn bộ chương trình, liên kết các module, gắn thêm các thủ tục PVM để cho phép chương trình chạy được trên một máy ảo song song.



Hình 2.5. Mô tả các giai đoạn của quá trình biên dịch

Ngoài ra để tăng hiệu quả của thuật toán song song, nhất là trên một máy ảo PVM (gồm nhiều bộ xử lý với năng lực khác nhau, tốc độ trao đổi dữ liệu khác nhau) phải cần đến một giải thuật nhằm tìm kiếm một lịch phân công phù hợp từng công việc cho mỗi bộ xử lý.

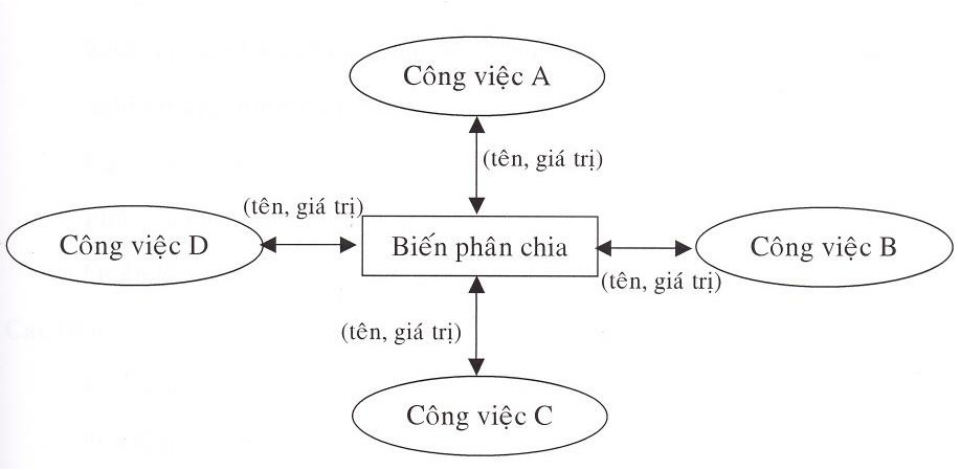
2.7.1. Quản lý biến phân chia

Một vấn đề cần đề cập trong lập trình song song là cơ chế truy nhập bộ nhớ phân chia. Trong đó các mô đun chạy đồng thời có thể dùng đọc / ghi giá trị tại một vùng nhớ.

Đây là một cơ chế cần thiết để các tiến trình có thể trao đổi dữ liệu với nhau. Thật không may máy ảo PVM (cũng như một số môi trường xử lý phân tán khác) không hỗ trợ cơ chế này.

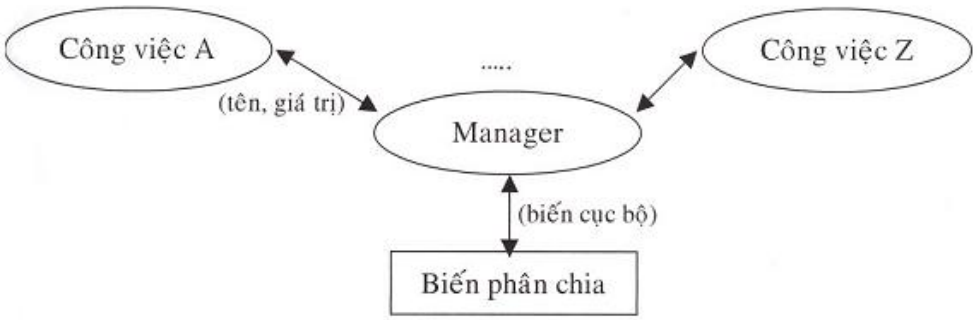
a. Mô hình quản lý biến phân chia

Trong máy tính, vùng nhớ được truy xuất thông qua địa chỉ vật lý. Địa chỉ này được quản lý bởi CPU phụ thuộc vào phần cứng. Trong khi đó, PVM lại không mô tả khái niệm về địa chỉ bộ nhớ, vì thế ta sẽ quản lý biến phân chia thông qua tên biến.



Hình 2.6. Mô hình quản lý biến phân chia

Để đảm bảo duy trì biến phân chia và sự truy nhập từ các công việc (task) của chương trình, ta cần có một tiến trình quản lý chung.



Hình 2.7. Mô hình quản lý tiến trình

b. Cơ chế hoạt động

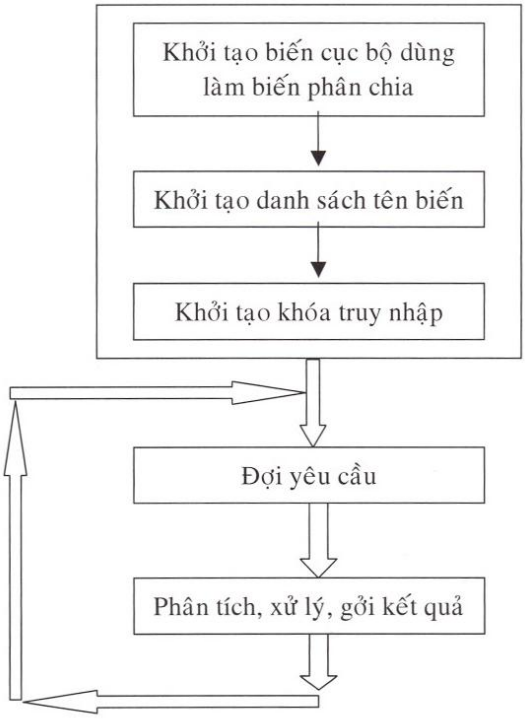
Chương trình quản lý biến phân chia thực hiện nhiệm vụ:

- Khởi tạo biến cục bộ làm thành biến phân chia.
- Khởi tạo một bảng băm lưu trữ danh sách tên biến phân chia được định nghĩa trong chương trình.
- Lắng nghe các yêu cầu truy xuất từ các tiến trình khác.
- Phục vụ các yêu cầu đồng thời.
- Quản lý việc truy nhập tương tranh.

Các tiến trình muốn truy nhập biến phân chia chỉ cần:

- Gửi một yêu cầu tới chương trình quản lý.
- Đọc/ghi giá trị biến phân chia mong muốn.

- Nhận kết quả phản hồi.



Hình 2.8. Mô hình cơ chế hoạt động

Một gói tin yêu cầu có cấu trúc:

- tid: Số hiệu công việc (task ID) của tiến trình gửi yêu cầu tới
- command: mã yêu cầu
- value: giá trị của biến (nếu có)

Điều khiển truy nhập tương tranh:

Khi có từ hai tiến trình trở lên cùng muốn truy nhập tới một biến phân chia, rất có thể giá trị của biến phân chia bị thay đổi không đúng với mong muốn của người lập trình vì thiếu sự đồng bộ giữa hai tiến trình.

Ví dụ có 2 task 1, 2 cùng truy nhập tới biến mảng A:

- Task 1 khởi tạo mảng A
- Task 2 đọc giá trị mảng A

Nếu không được đồng bộ, rất có thể task 2 sẽ nhận được mảng A chỉ có một phần đầu được khởi tạo.

Để khắc phục ta sẽ sử dụng cơ chế khóa ngoại trừ:

Mỗi biến được gắn một khóa và các thủ tục nguyên thủy (primitive)

- lock_sh_var( ) : khóa một biến phân chia bằng tiến trình hiện thời
- unlock_sh_var( ) : mở khóa. Chỉ có tiến trình khóa biến thì mới mở được
- get_sh_ var( ) : đọc giá trị của biến
- put_sh_var( ) : lưu giá trị mới vào biến
- get_lock_sh_var( ) : đọc giá trị của biến và khoá lại
- put_unlock_sh_var( ) : lưu giá trị mới vào biến và mở khóa

Cơ chế hoạt động của khóa ngoại trừ:

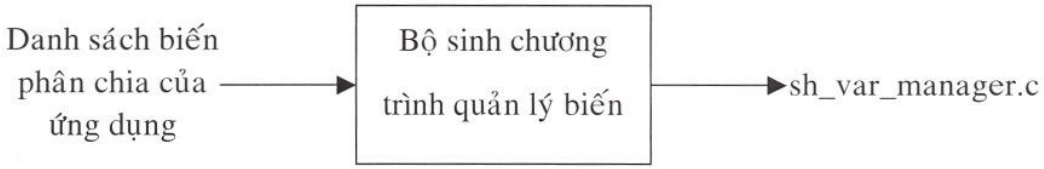
- Một biến phân chia khi bị khóa bởi một tiến trình nào thì chỉ có thể được mở bằng tiến trình đó.
- Khi một biến khóa thì mọi truy nhập từ các tiến tình khác sẽ phải chờ đợi (block) cho đến khi biến được mở khóa.

Thiết kế chương trình quản lý biến phân chia trong môi trường PVM:

Trước hết, người lập trình cung cấp một danh sách các biến phân chia trong ứng dụng. Mỗi biến gồm các thông tin về:

- Kiểu dữ liệu cơ bản.
- Tên biến.
- Kích thước nếu là biến mảng.

Để đảm bảo tính nhất quán, với mỗi ứng dụng, ta sinh ra một chương trình quản lý biến phân chia riêng.



Hình 2.9. Quản lý biến phân chia

Cấu trúc của mỗi phần tử trong danh sách biến phân chia:

```
struct {
        int type ; Kiểu dữ liệu cơ bản của biến: int, long, float, double...
        char * name ; tên biến.
```

int size ; size = 0: biến đơn.

size > 0: biến mảng.

} ShVarType;

Để đảm bảo chỉ có những mô đun thuộc ứng dụng mới truy nhập được các biến phân chia, ta tạo ra một nhóm tiến trình trong máy ảo PVM duy nhất đối với mỗi ứng dụng.

Thuật toán chi tiết để quản lý biến phân chia:

- Bước 1: Khởi tạo
  - Khởi tạo các biến cục bộ tương ứng với biến phân chia.
  - Khởi tạo bảng băm chứa các tên biến truy nhập có đưa thêm khóa ngoại trừ.
  - Khởi tạo hàng đợi yêu cầu
- Bước 2: Gia nhập nhóm
- Bước 3: Đợi lệnh gửi tới
- Bước 4: Phân tích lệnh
- Bước 5: Thực thi lệnh
- Bước 6: Nhận được lệnh kết thúc không?
  - Nếu đúng sang bước 7
  - Ngược lại quay bước 2
- Bước 7: Dọn dẹp trước khi kết thúc.

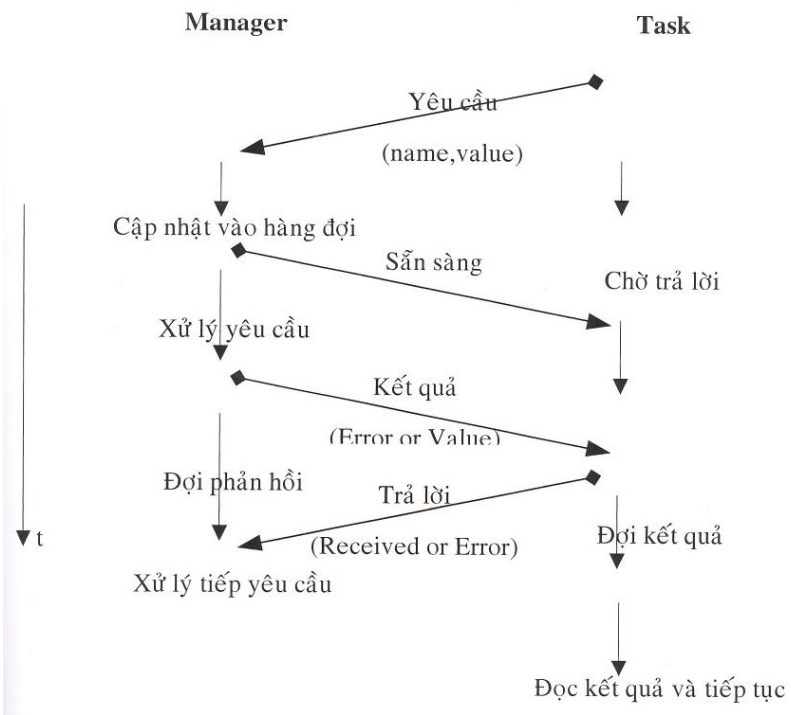
Các lệnh gồm có:

- command ==0 : Lệnh kết thúc chương trình quản lý
- command == 1 : Lệnh đọc giá trị của biến phân chia
- command == 2 : Lệnh lưu giá trị vào biến phân chia
- command == 3 : Khóa một biến
- command == 4 : Mở khóa
- command == 5 : Đọc giá trị của khóa. Xác định tiến trình nào đang khóa biến.

- command == 11 : Đọc và khóa biến phân chia
- command == 22 : Ghi giá trị mới và mở khóa
- command == 12 : Đọc và mở khóa
- command == 21 : Ghi và khóa

Giao thức truyền thông giữa các tiến trình (task) và tiến trình quản lý biến phân chia:

- Các task gửi gói yêu cầu tới tiến trình quản lý
- Tiến trình quản lý sắp xếp yêu cầu trong một hàng dợi để đảm bảo thứ tự và không làm thất lạc các gói yêu cầu.
- Tiến trình quản lý lấy một yêu cầu trong hàng đợi gửi trả lại thông điệp báo " sẵn sàng" (ready packet)
- Sau khi xử lý xong yêu cầu, tiến trình quản lý gửi trả lại gói kết quả cho task yêu cầu.
- Task yêu cầu nhận kết quả, sau đó gửi thông điệp báo kết thúc phiên yêu cầu.



Hình 2.10. Giao thức truyền thông

2.7.2. Giao diện với người lập trình

Các phần trên đã trình bày lần lượt các chức năng cơ bản bên trong của một số môi trường lập trình song song. Ngoài ra, còn một phần khá quan trọng để giúp mô tả dễ dàng đồ thị công việc, ràng buộc vào/ra, ..., đó là phần giao diện.

Phần giao diện này còn có đòi hỏi phải được cài đặt trên mạng máy tính cho đồng thời nhiều người lập trình cùng làm việc.

Nhiệm vụ của phần giao diện:

- Liên kết với một chương trình quản lý chung trên máy chủ
- Bộ soạn thảo mã nguồn cho mỗi công việc
- Quản lý từ xa máy ảo song song PVM
- Biên dịch chạy thử chương trình và hiển thị kết quả

Chương trình quản lý môi trường trên máy chủ có nhiệm vụ:

- Liên lạc với các giao diện.
- Quản lý các phiên làm việc của người lập trình.
- Quản lý lưu trữ mã nguồn của các công việc ứng với mỗi ứng dụng.
- Quản lý máy ảo PVM.
- Gọi chạy các ứng dụng trên PVM, chuyển kết quả tới giao diện.

CHƯƠNG 3: THỰC NGHIỆM

3.1. Phát biểu bài toán

Chương này ta sẽ tìm hiểu về cách thức chạy một bài toán đơn giản. Do thời gian và kiến thức có hạn nên ta sẽ tính toán một bài toán thử nghiệm. PVM sẽ được cài trên môi trường máy ảo LINUX và chạy trên giao diện dòng lệnh. Bài toán sẽ in ra màn hình các kết quả từ các máy và định danh của các máy.

3.2. Xây dựng các toán tử trong bài toán

- Tư tưởng giải quyết thuật toán:
  - Số tiến trình của bài toán sẽ gấp 3 lần số máy tham gia hệ thống.
  - Máy Master có nhiệm vụ gửi dữ liệu cho các máy Slave để tính toán.
  - Master nhận kết quả từ các máy Slave.
  - In ra kết quả của từng máy và định danh của các máy đó.
- Song song hóa giải thuật tuần tự:

Sử dụng hệ thống gồm NPROCS bộ xử lý, ở đây NPROCS = 3 * nhost. Khi đó mỗi bộ xử lý sẽ phải tính kết quả theo công thức và trả kết quả về máy master. Lúc này máy chủ sẽ tổng hợp và in ra màn hình các kết quả đó.

- Chương trình master1.c:

```
#include <stdio.h>
#ifdef HASSTDLIB
#include <stdlib.h>
#endif
#include "pvm3.h"
#define SLAVENAME "slave1"
main()
{
      int mytid;                  /* my task id */
      int tids[32];                     /* slave task ids */
      int n, nproc, numt, i, who, msgtype, nhost, narch;
      float data[100], result[32];
      struct pvmhostinfo *hostp;
      /*Ghi danh vào PVM*/
      mytid = pvm_mytid();

      /* Thiết lập số máy slave để bắt đầu */
      pvm_config( &nhost, &narch, &hostp );
      nproc = nhost * 3;
      if( nproc > 32 ) nproc = 32 ;
      printf("Spawning %d worker tasks ... " , nproc);
      /* Bắt đầu với các tác vụ của slave */
      numt = pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
```

```
      if( numt < nproc )
      {
      printf("\n Trouble spawning slaves. Aborting. Error codes
      are:\n");
      for( i=numt ; i<nproc ; i++ )
       {
          printf("TID %d %d\n",i,tids[i]);
       }
      for( i=0 ; i<numt ; i++ )
      {
          pvm_kill( tids[i] );
       }
       pvm_exit();
       exit(1);
   }
     printf("SUCCESSFUL\n");
    /* Chương trình người dùng*/
    n = 100;
    /* Khởi tạo dữ liệu( data, n ); */
    for( i=0 ; i<n ; i++ ){
       data[i] = 1.0;
    }
    /* Gửi dữ liệu ban đầu cho các tác vụ slave */
     pvm_initsend(PvmDataDefault);
     pvm_pkint(&nproc, 1, 1);
     pvm_pkint(tids, nproc, 1);
     pvm_pkint(&n, 1, 1);
     pvm_pkfloat(data, n, 1);
     pvm_mcast(tids, nproc, 0);
     /* Đợi các kết quả từ slave */
      msgtype = 5;
      for( i=0 ; i<nproc ; i++ )
     {
      pvm_recv( -1, msgtype );
      pvm_upkint( &who, 1, 1 );
      pvm_upkfloat( &result[who], 1, 1 );
      printf("I got %f from %d; ",result[who],who);
      if (who == 0)
                printf( "(expecting %f)\n", (nproc - 1) * 100.0);
      else
                printf( "(expecting %f)\n", (2 * who - 1) *
          100.0);
      }
       /* Thoát chương trình PVM trước khi dừng */
    pvm_exit();
}
```

- Chương trình slave1.c

```
#include <stdio.h>
#include "pvm3.h"
main()
{
    int mytid;        /* my task id */
    int tids[32];     /* task ids   */
    int n, me, i, nproc, master, msgtype;
```

```
    float data[100], result;
    float work();
    /* Ghi danh vào PVM */
     mytid = pvm_mytid();
    /* Nhận dữ liệu từ master */
     msgtype = 0;
     pvm_recv( -1, msgtype );
      pvm_upkint(&nproc, 1, 1);
      pvm_upkint(tids, nproc, 1);
      pvm_upkint(&n, 1, 1);
      pvm_upkfloat(data, n, 1);
    /* Xác định định danh của slave(0 -- nproc-1) */
    for( i=0; i<nproc ; i++ )
       if( mytid == tids[i] ){ me = i; break; }
    /* Tính toán dữ liệu */
    result = work( me, n, data, tids, nproc );
    /* Gửi kết quả đến master */
    pvm_initsend( PvmDataDefault );
    pvm_pkint( &me, 1, 1 );
    pvm_pkfloat( &result, 1, 1 );
    msgtype = 5;
    master = pvm_parent();
    pvm_send( master, msgtype );
    /* Chương trình kết thúc. Thoát PVM trước khi dừng*/
    pvm_exit();
}
float
work(me, n, data, tids, nproc )
    /* Một ví dụ đơn giản: các máy slave trao đổi dữ liệu với các
slave bên trái*/
    int me, n, *tids, nproc;
    float *data;
{
    int i, dest;
    float psum = 0.0;
    float sum = 0.0;
    for( i=0 ; i<n ; i++ ){
       sum += me * data[i];
    }
    /* Minh họa giao tiếp node – to - node */
    pvm_initsend( PvmDataDefault );
    pvm_pkfloat( &sum, 1, 1 );
    dest = me+1;
    if( dest == nproc ) dest = 0;
    pvm_send( tids[dest], 22 );
    pvm_recv( -1, 22 );
    pvm_upkfloat( &psum, 1, 1 );

    return( sum+psum );
}
```

3.3. Cài đặt và đánh giá

3.3.1. Cấu hình hệ thống

Để cài đặt PVM trên hệ điều hành LINUX thì trước tiên ta phải cấu hình được hệ thống. Cấu hình trên 2 node Master và Slave.

- Với cả 2 node:
  - Tạo một user bất kỳ dùng chung cho 2 node Master và Slave. Khởi động và tắt các dịch vụ không cần thiết của hệ thống, tắt FireWall,…bằng lệnh setup.
  - Đặt hostname và IP cho các node trong tập tin /etc/hosts để các node ssh vào nhau thông qua hostname của 2 máy thay cho địa chỉ IP.
  - Chỉnh sửa tập tin /etc/hosts.equiv để kết nối rsh vào nhau mà không yêu cầu mật khẩu (PVM yêu cầu cần phải có file này). Nếu chưa có file này thì cần phải tạo mới bằng trình vi. Nội dung của file này là địa chỉ IP của 2 máy.
  - Tạo tập tin ẩn .rhosts trên thư mục $HOME của user. Tập tin này cũng cho phép rsh giữa các node không cần mật khẩu và nội dung của file này bao gồm hostname của các node.
  - Thêm các lệnh rsh, rexec, rlogin vào cuối tập tin /etc/securetty.
- Đối với máy master:
  - Đặt lại địa chỉ IP tĩnh như trong khai báo của tập tin hosts cho node master.
  - Đặt lại tên máy master trong tập tin /etc/sysconfig/network.
  - Cấu hình lại tập tin /etc/exports để tạo thư mục dùng chung cho hệ thống. Cụ thể node Master sẽ export thư mục home dùng chung, khi đó node Slave sẽ mount được các thư mục có trong /home của Master. Thêm các dòng lệnh vào cuối file /etc/exports:

    ```
    /home          192.168.1.0/255.255.255.0(rw)
    ```

  - Tạo key và cấp quyền cho ssh trên thư mục $HOME của user:

    ```
    $ssh-keygen –t rsa
    $cd $HOME/.ssh
    $cp id_rsa.pub authorized_keys
    ```

$chmod 600 *

$chmod 755 .ssh

- Đối với máy Slave:
  - Đặt lại địa chỉ IP và tên máy.
  - Cấu hình tập tin /etc/fstab để mount thư mục public trên node Master về. Cụ thể, thêm dòng lệnh vào cuối file /etc/fstab:

    master:/home /home nfs rw 0 0

3.3.2. Cài đặt PVM

Phần này đã được trình bày ở mục trước. Tuy nhiên, sau khi tải PVM về cần đặt biến môi trường để PVM có thể chạy được.

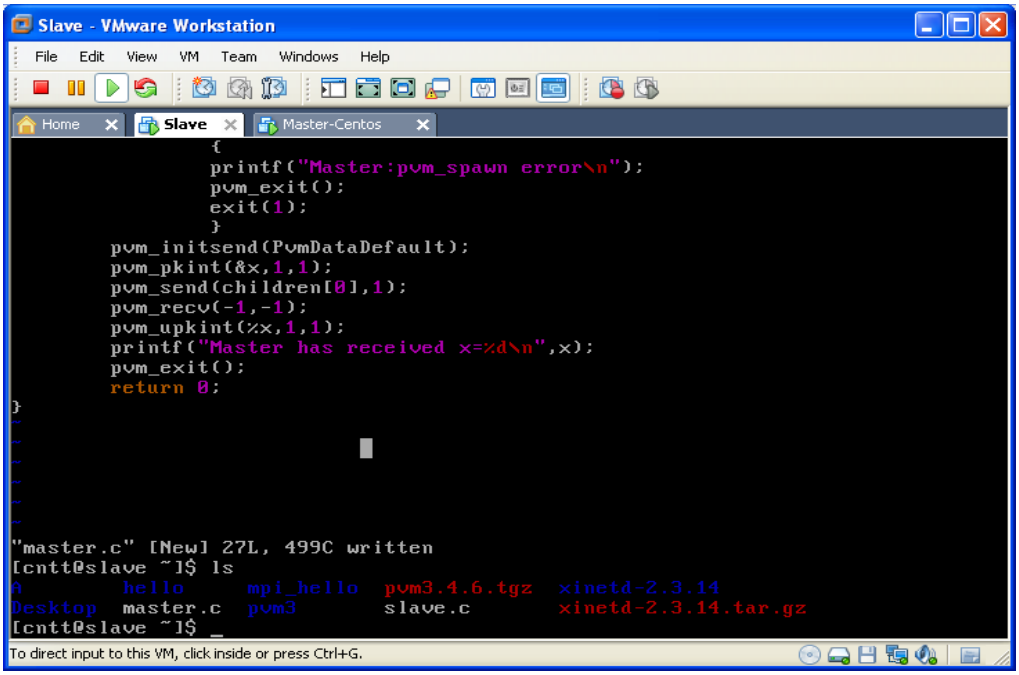
3.3.3. Biên dịch và chạy thử

Khởi động PVM bằng lệnh $PVM, sau đó add host slave vào và chạy chương trình.

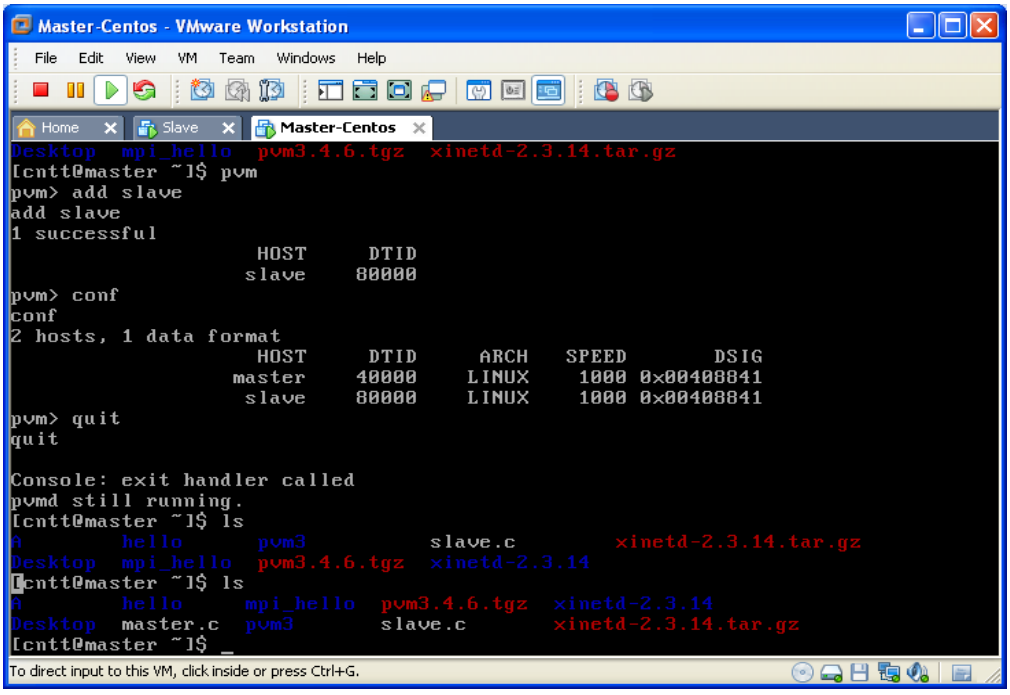
Vào thư mục $HOME/pvm3/examples và biên dịch file master1.c và slave1.c:

- $cd $HOME/pvm3/examples
- $aimk master1 slave1
- Kết quả file master và worker được tạo trong thư mục $PVM_ROOT/bin/LINUX. Vào thư mục này và thực thi file master1.
- $cd $PVM_ROOT/bin/LINUX
- $./master1 và kết quả bài toán ở hình 3.3.

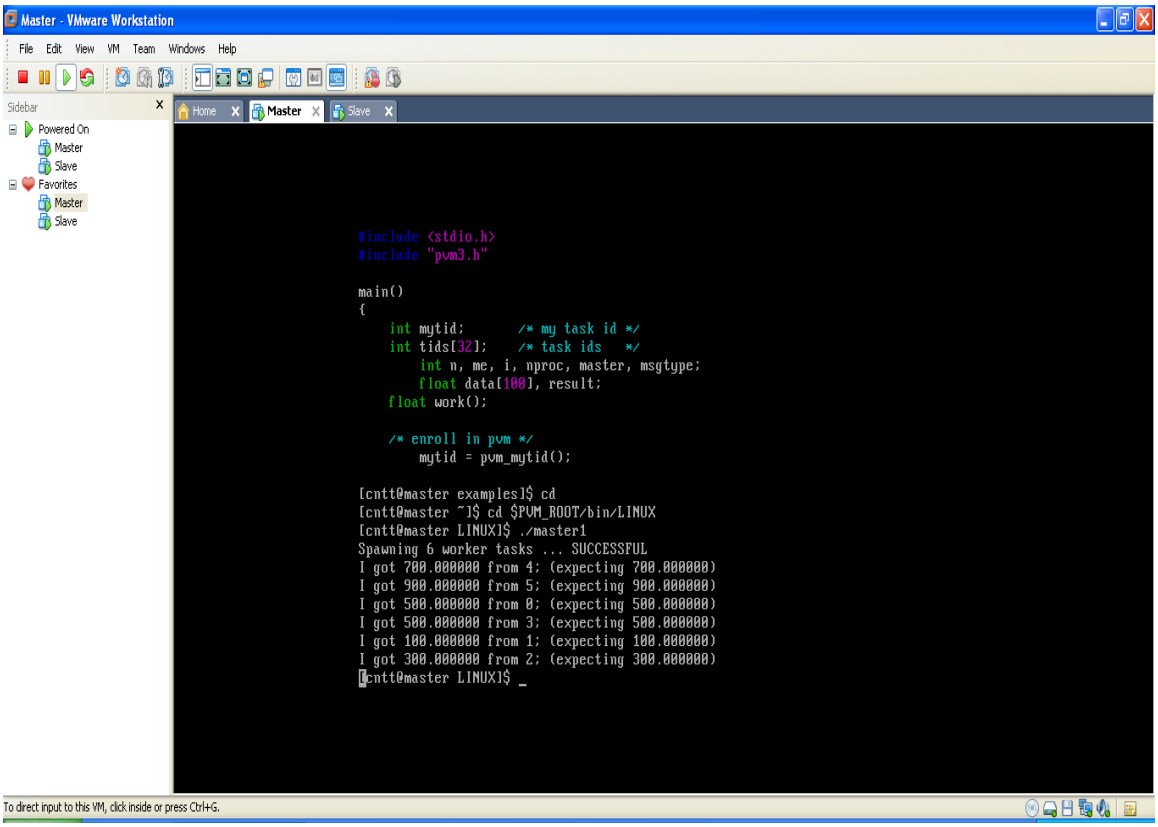
3.3.4. Kết quả



Hình 3.1. Node Slave đã mount được thư mục /home của node Master



Hình 3.2. PVM đã được cài và đã add host Slave



Hình 3.3. Kết quả của bài toán

3.3.5. Đánh giá

Hình 3.3 đã đưa ra kết quả của bài toán. Tuy nhiên trên thực tế, để cấu hình được 2 node thì cần nhiều thời gian. Do đó để có thể cấu hình được các node một cách nhanh nhất thì phải thực hiện các bước một cách chính xác.

Áp dụng giải thuật song song với bài toán tính toán, ta thấy có sự khác biệt rõ ràng về mặt thời gian. Thay vì việc tính lần lượt các phép tính trên cùng một máy, chia nhỏ bài toán thành từng phần để các máy cùng xử lý sẽ tiết kiệm và tận dụng được chi phí tính toán. Tuy bài toán còn nhỏ, sự khác biệt chưa lớn nhưng sẽ là tiền đề để phát triển được những bài toán lớn hơn.

KẾT LUẬN

Trong khuôn khổ của bài khóa luận, em đã nghiên cứu, tìm hiểu các kiến thức cơ bản nhất về xử lý song song và phân tán, qua đó có thể nắm được các nguyên tắc, cấu trúc, cơ chế và phương pháp thực hiện song song để phát triển những phần mềm khai thác hiệu quả những khả năng tính toán song song, phân tán của các máy tính hiện đại nhằm giải quyết các bài toán đặt ra trong thực tế. Bên cạnh đó, em cũng đã tìm hiểu được lập trình trên máy ảo song song với PVM bằng ngôn ngữ C/C++, cài PVM trên máy ảo của hệ điều hành LINUX và chạy một ví dụ ứng dụng.

Tuy nhiên, vấn đề này có liên quan trực tiếp đến kiến trúc của máy tính, phần mềm hệ thống (hệ điều hành), ngôn ngữ lập trình,…đặc biệt là làm việc và cấu hình hệ thống trên giao diện dòng lệnh của hệ điều hành LINUX nên khá mới mẻ và gặp nhiều khó khăn.

Với sự phát triển của công nghệ thông tin, tính toán phân tán và song song sẽ tiến đến nghiên cứu sâu hơn để có thể áp dụng giải được các bài toán lớn và phức tạp trong thực tế.

Mặc dù đã rất cố gắng nhưng trong quá trình tìm hiểu, ngoài một số kiến thức đã tìm hiểu được, bài khóa luận còn nhiều hạn chế và thiếu sót mà em chưa thể tìm hiểu và cập nhật kịp thời. Kính mong các thầy cô chỉ bảo và giúp đỡ để em hoàn thành bài khóa luận này.

Em xin chân thành cảm ơn!

Tài liệu tham khảo

Tài liệu tiếng Việt:

[1]. PGS. TS. Đoàn Văn Ban, TS. Nguyễn Mậu Hân - **Xử lý song song và phân tán** – Nhà xuất bản khoa học và kỹ thuật, 2006.

Tài liệu tiếng Anh:

[2]. Al Geist, Adam Beguelin, Jack Dongarra, Weichang Jiang, Robert Manchek, Vaidy Sunderam - **PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing/**, - London: The MIT Press, 1995.

[3]. Barry Wilkingson, Michael Allen – **Parallel Programming, Technigues and Applications Using Networked Workstations and Parallel Computers** / Prentice Hall New Jersey, 1999.

[4]. http://www.csm.ornl.gov/pvm/

[5]. http://www.netlib.org/pvm3/book/pvm3-book.html

[6]. http://www.ontko.com/pub/rayo/cs40/pvm.html

[7]. http://www.parallels.com/