BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC ĐÀ NẴNG

NGUYỄN TẤN THẮNG

THUẬT TOÁN SONG SONG GIẢI QUYẾT MỘT SỐ BÀI TOÁN VỀ LÝ THUYẾT ĐỒ THỊ

Chuyên ngành: KHOA HỌC MÁY TÍNH
Mã số: 60.48.01

TÓM TẮT LUẬN VĂN THẠC SĨ KỸ THUẬT

Đà Nẵng - Năm 2011

Công trình được hoàn thành tại
ĐẠI HỌC ĐÀ NẴNG

Người hướng dẫn khoa học: **PGS.TSKH Trần Quốc Chiến**

Phản biện 1:

Phản biện 2:

Luận văn sẽ được bảo vệ trước Hội đồng chấm Luận văn tốt nghiệp thạc sĩ kỹ thuậttính họp tại Đại học Đà Nẵng vào ngày ….. tháng 10 năm 2011

** Có thể tìm hiểu luận văn tại:*

- Trung tâm Thông tin - Học liệu, Đại học Đà Nẵng
- Trung tâm học liệu, Đại học Đà Nẵng.

MỞ ĐẦU

1. Lý do chọn đề tài:

Khoa học kỹ thuật ngày càng phát triển, đặt ra nhiều bài toán với khối lượng tính toán rất lớn. Trong số đó có những bài toán mà kết quả chỉ có ý nghĩa nếu được hoàn thành trong khoảng thời gian cho phép. Ví dụ như các tính toán trong thời gian thực, mô phỏng sự chuyển động của các phân tử, tính quĩ đạo chuyển động của vật thể trong không gian, dự báo thời tiết...

Để giải quyết những bài toán này, người ta đã nghiên cứu tăng tốc độ tính toán bằng hai phương pháp hay kết hợp cả hai:

Phương pháp 1: Cải tiến công nghệ, tăng tốc độ xử lý của máy tính. Công việc này đòi hỏi nhiều thời gian, công sức và tiền của, nhưng tốc độ cũng chỉ đạt được đến một giới hạn nào đó.

Phương pháp 2: Chia bài toán ra thành những công việc nhỏ để có thể chạy song song trên nhiều bộ xử lý.

Việc phát triển công nghệ tính toán theo phương pháp 2 đã cho ra đời công nghệ tính toán song song, đó là việc sử dụng đồng thời nhiều tài nguyên tính toán để giải quyết một bài toán. Các tài nguyên tính toán có thể bao gồm một máy tính với nhiều bộ vi xử lý hay một tập các máy tính kết nối mạng hay là một sự kết hợp của hai dạng trên. Công nghệ tính toán song song cho phép giảm thời gian thực thi bài toán tùy thuộc cách phân chia và số bộ xử lý thực thi chương trình. Nguyên tắc quan trọng nhất của tính toán song song chính là tính đồng thời hay xử lý nhiều tác vụ cùng một lúc.

Trong tính toán song song hiện nay, có hai công nghệ chính:

Thứ nhất là sử dụng các siêu máy tính với rất nhiều bộ xử lý được tích hợp bên trong được thiết kế đồng bộ cả về phần cứng và phần mềm. Các công nghệ được áp dụng trong các siêu máy tính thường là các công nghệ tiên tiến làm cho giá thành của hệ thống siêu máy tính tăng rất cao.Vì

thế các siêu máy tính thường được sử dụng trong các lĩnh vực mà vấn đề tính toán phức tạp, nhạy cảm và yêu cầu thời gian thực như mô phỏng thực hiện của các động cơ máy bay, quốc phòng, vũ trụ...

Cách thứ hai là kết nối các máy tính lại với nhau và cùng thực hiện bài toán. Hệ thống các máy tính kết nối này chính là hệ thống tính toán song song phân cụm. Hệ thống này có ưu điểm là giá thành rẻ hơn rất nhiều so với siêu máy tính có cùng sức mạnh (do sử dụng các thiết bị thông thường) và tính linh hoạt của hệ thống (số nút, số bộ xử lý, bộ nhớ, thiết bị mạng... đều mang tính tuỳ biến cao). Sự phát triển mạnh mẽ của mạng máy tính, các công nghệ mạng hiện nay đã lấp đi hạn chế về truyền thông trong hệ thống máy tính song song phân cụm làm cho nó được phát triển rộng rãi. Các lĩnh vực sử dụng hệ thống tính toán song song phân cụm thường yêu cầu tính toán không quá lớn, không yêu cầu thời gian thực như xử lý ảnh, nhận dạng vân tay, tính toán kết cấu công trình, mô phỏng các thí nghiệm...

Với sự ra đời của chíp đa lõi (multi-core) và xử lí đa luồng (multi-threads) thì việc khai thác hết khả năng xử lí của nó là một vấn đề cần quan tâm hiện nay. Tuy nhiên với lập trình truyền thống (lập trình tuần tự), các câu lệnh, các quá trình xử lý được thực hiện một cách lần lượt, tuần tự như vậy sẽ không phát huy hết hiệu năng của bộ vi xử lý đa lõi. Các nhà sản xuất chip và phần mềm máy tính đã bắt đầu những nỗ lực để định hướng giới phát triển phần mềm, cung cấp cho họ những công cụ tốt hơn trong việc lập trình đa lõi. Điều đó có nghĩa là người lập trình phải lập trình lại theo một cách khác để tận dụng sự gia tăng về số lõi.

Với mục đích tìm hiểu và nghiên cứu về thuật toán song song, tôi chọn đề tài “Thuật toán song song cho một số bài toán về lí thuyết đồ thị” nhằm tìm hiểu, nghiên cứu và tìm giải pháp song song cho một số bài toán về lý thuyết đồ thị trên cơ sở những thuật toán tuần tự truyền thống.

2. Mục tiêu của đề tài:

Tìm hiểu, nghiên cứu và xây dựng thuật toán song song cho một số bài toán cụ thể để nâng cao khả năng thực hiện của chương trình, giảm thời gian thực hiện, góp phần năng cao hiệu năng hoạt động của hệ thống.

3. Đối tượng và phạm vi nghiên cứu:

+ Đối tượng nghiên cứu:

- XLSS và thuật toán song song.

- Lý thuyết đồ thị.

+ Phạm vi nghiên cứu:

- Một số bài toán về lý thuyết đồ thị: Tìm đường đi, cây bao trùm.

4. Phương pháp nghiên cứu:

- Nghiên cứu lý thuyết: XLSS, thuật toán song song, lý thuyết đồ thị.
- Tìm hiểu một số thuật toán cơ bản: Tìm đường đi, cây bao trùm.
- Nghiên cứu và xây dựng thuật toán song song.

4.1 Ý nghĩa khoa học và thực tiễn của đề tài:

- Nghiên cứu và tìm giải pháp nâng cao hiệu năng của hệ thống bằng XLSS.
- Có thể áp dụng cho một số lĩnh vực cụ thể và một số bài toán có độ phức tạp về thời gian lớn, những bài toán thời gian thực.

***4.2 Bố cục của đề tài*:**

Nội dung của đề tài này bao gồm 3 chương:

Chương 1: Tính toán song song và thuật toán song song.

- Chương này giới thiệu tổng quan về tính toán song song, thuật toán song song, các mô hình lập trình song song.

Chương 2: Lí thuyết đồ thị.

- Chương này giới thiệu tổng quan về lí thuyết đồ thị.

Chương 3: Song song hóa một số bài toán về lí thuyết đồ thị

- Chương này phát biểu, mô tả và thực hiện song song hóa bài toán tìm cây bao trùm nhỏ nhất và bài toán tìm đường đi ngắn nhất từ đỉnh nguồn đến mọi đỉnh trên đồ thị có trọng số.

Kết luận: Nêu lên những vấn đề đã nghiên cứu và kết quả đạt được, những hạn chế và hướng phát triển của đề tài.

CHƯƠNG 1
ĐẠI CƯƠNG VỀ XỬ LÍ SONG SONG

1.1 Một số khái niệm về xử lí song song

Xử lí song song là cách xử lí thông tin bằng việc sử dụng nhiều hơn một bộ xử lí để thực hiện nhiều hơn một thao tác trên dữ liệu tại một thời điểm.

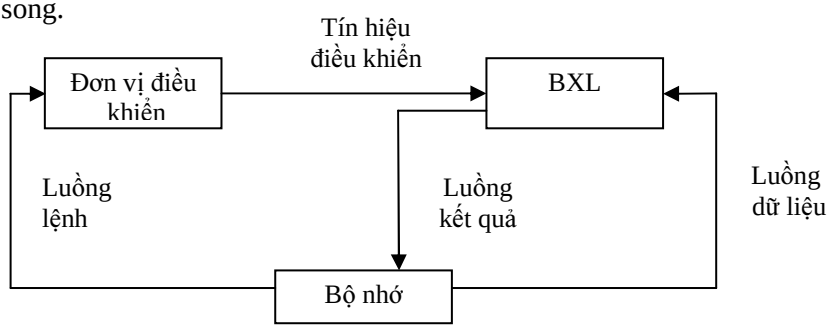
Thuật toán song song là một tập các tiến trình (process) hoặc các tác vụ (task) có thể thực hiện đồng thời và có thể trao đổi dữ liệu với nhau để kết hợp cùng giải một bài toán đặt ra.

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

1.2 Các kiến trúc song song

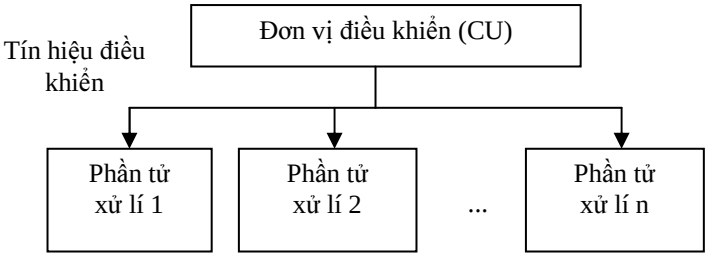
1.2.1 Mô hình SISD : Đơn lệnh, đơn dữ liệu

Máy tính theo mô hình SISD 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, 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. Hay nói cách khác, máy tính loại SISD là máy tính thông thường, chỉ có duy nhất một bộ vi xử lý, không có cấu trúc song song và cũng không có dữ liệu song song.



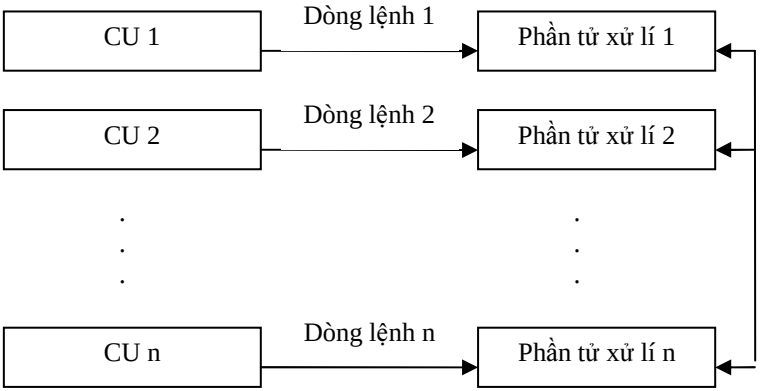
1.2.2 Mô hình SIMD: Đơn lệnh, đa dữ liệu

Máy theo mô hình 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. Đơn vị điều khiển (CU) phát sinh tín hiệu điều khiển tới tất cả các phần tử xử lý, những bộ xử lí này cùng thực hiện một phép toán trên các mục dữ liệu khác nhau, nghĩa là mỗi bộ xử lí có luồng dữ liệu riêng.



1.2.3 Mô hình MISD: Đa lệnh, đơn dữ liệu

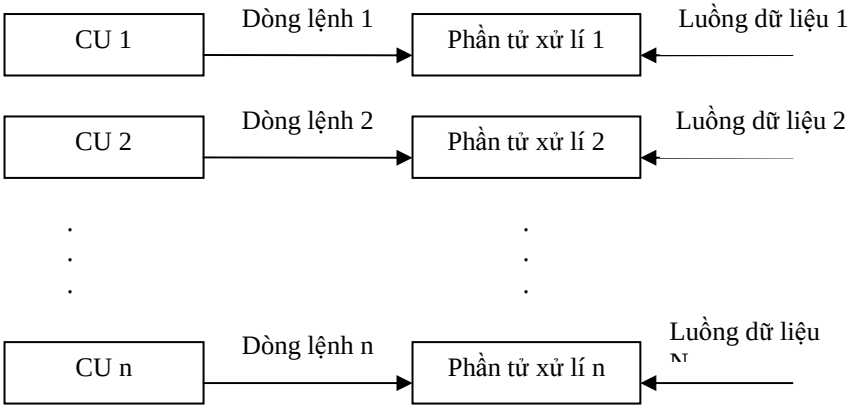
Máy tính loại MISD là ngược lại với SIMD. Máy tính MISD có thể thực hiện nhiều chương trình (nhiều lệnh) trên cùng một mục dữ liệu.



1.2.4 Mô hình MIMD: Đa lệnh, đa dữ liệu

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

liệu riêng.



1.3 Thiết kế và đánh giá thuật toán song song

1.3.1 Nguyên lý thiết thiết kế thuật toán song song

Khi muốn thực hiện việc xử lí song song ta phải xét cả kiến trúc máy tính và các 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 lí).
- Các tiến trình được đồng bộ ra sao

Khi thiết kế một thuật toán song song có thể sử dụng năm nguyên lí chính trong thiết kế thuật toán song song:

+ Nguyên lý lập lịch: mục đích là giảm tối thiểu các bộ xử lí 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 $\{T_1, T_2, \ldots, T_n\}$, trong đó T_{i+1} thực hiện sau khi T_i 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 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.

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. Khi chuyển một thuật toán tuần tự sang thuật toán song song hoặc chuyển một thuật toán song song thích hợp với kiến trúc đang có. Cần xác định được yêu cầu sau:

- *Kiến trúc tính toán nào sẽ phù hợp với bài toán?*

- *Những bài toán loại nào sẽ xử lý hiệu quả trong kiến trúc song song cho trước ?*

1.3.2 Các giai đoạn thiết kế thuật toán song song

- Song song hóa các thuật toán tuần tự, biến đổi những cấu trúc tuần tự để tận dụng khả năng song song tự nhiên của tất cả các thành phần trong hệ thống xử lí.

- Thiết kế thuật toán song song hoàn toàn mới.

- Thiết kế thuật toán song song từ những thuật toán song song đã được xây dựng.

1.3.3 Đánh giá thuật toán song song

+ Thời gian tính toán

Thời gian tính toán của giải thuật song song là thời gian dành để thực hiện tính toán. Thời gian tính toán sẽ phụ thuộc vào số tác vụ thực hiện.

+ Thời gian truyền thông

Thời gian truyền thông của một giải thuật là thời gian các tác vụ dành

để gửi và nhận thông điệp.

+ Thời gian rỗi

Một BXL có thể đặt trong trạng thái rỗi nếu thiếu tính toán hoặc dữ liệu để tính toán.

+ Tốc độ và hiệu quả:

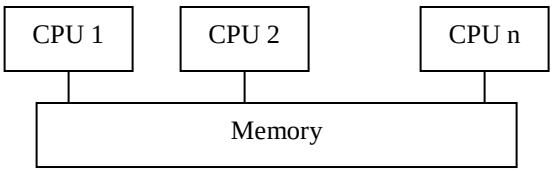
Hiệu quả của thuật toán được định nghĩa là phần thời gian mà các bộ xử lí dùng để thực hiện công việc có ích, chỉ ra mức độ hiệu quả của một giải thuật khi sử dụng các tài nguyên tính toán của một chương trình song song theo hướng độc lập với kích thước bài toán.

Thuật toán song song có thể có độ phức tạp lớn hơn thuật toán tuần tự, do đó rất khó để đánh giá thuật toán song song. Kết quả đạt được thường được đánh giá bằng thực nghiệm chương trình.

1.4 Một số mô hình lập trình song song

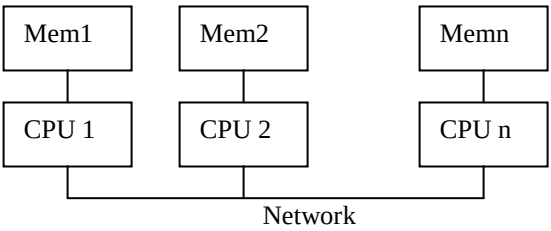
1.4.1 Mô hình chia sẻ bộ nhớ:

Trong mô hình này các BXL đều có thể truy cập vào bộ nhớ chia sẻ, các BXL có thể hoạt động độc lập nhưng luôn chia sẻ địa chỉ các ô nhớ.



1.4.2 Mô hình truyền thông điệp

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: gửi và nhận thông điệp.



1.5 Môi trường lập trình song song

1.5.1 MPI

MPI là viết tắt của Message Passing Interface. Đó là một thư viện các hàm trong C và Fortran cho phép chèn vào trong code để thực hiện trao đổi dữ liệu giữa các tiến trình.

MPI thường sử dụng cho hệ thống có kiến trúc bộ nhớ phân tán (hệ thống máy tính phân cụm (cluster))), tuy nhiên nó cũng hoạt động bình thường trên hệ thống kiến trúc chia sẻ bộ nhớ.

Chương trình MPI được dịch và chạy trên nền tảng có hỗ trợ chuẩn MPI.

Trong mô hình lập trình MPI, các tiến trình truyền thông bằng cách gọi các hàm thư viện để gửi và nhận thông điệp với những tiến trình khác. Trong hầu hết các chương trình MPI, số bộ xử lí cố định khi khởi tạo chương trình, và một tiến trình được tạo ra trên một bộ xử lí. Tuy nhiên, những tiến trình này có thể chạy những chương trình khác nhau.

Một chương trình MPI bao gồm nhiều chương trình tuần tự có trao đổi dữ liệu với nhau thông qua việc gọi các hàm trong thư viện.

1.5.2 OpenMP

OpenMP là công cụ cho phép lập trình song song hỗ trợ C/C++ và Fortran77/90. OpenMP hoạt động trên hệ thống kiến trúc chia sẻ bộ nhớ và các máy tính đa lõi (multi-core).

OpenMP cung cấp mô hình lập trình đa luồng cấp cao, xây dựng trên thư viện lập trình đa luồng của hệ thống. Theo mô hình này người lập trình có thể tạo nhóm các luồng cho thực hiện song song đồng thời chỉ rõ cách các chia sẻ công việc giữa các luồng thành viên của nhóm. Cho phép khai báo dữ liệu chia sẻ, riêng tư và đồng bộ các luồng, cho phép các luồng thực

hiện thực hiện công việc một các độc quyền. Ngoài ra OpenMP còn hỗ trợ quản lí thời gian thực hiện và số lượng luồng.

OpenMP cho phép viết chương trình song song trong khi vẫn giữ nguyên mã nguồn của chương trình tuần tự. OpenMP sử dụng chỉ thị chương trình dịch để điều khiển sự song song.

OpenMP đưa ra 2 cách song song đó là:

+ Song song theo chức năng: lập trình song song có cấu trúc dựa trên phân chia công việc trong vòng lặp.

+ Song song theo dữ liệu: Hỗ trợ việc gán các công việc cụ thể cho các luồng thông qua chỉ số của luồng.

CHƯƠNG 2

ĐẠI CƯƠNG VỀ ĐỒ THỊ

2.1 Các khái niệm cơ bản về đồ thị

2.1.1 Định nghĩa đồ thị

Định nghĩa 2.1: Một đơn đồ thị vô hướng là một bộ G=(V,E), trong đó:

- V ≠∅ là tập hợp hữu hạn gồm các đỉnh của đồ thị.
- E là tập hợp các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.

Định nghĩa 2.2: Đơn đồ thị có hướng là một bộ G=(V,E), trong đó:

- V ≠∅ là tập hợp hữu hạn gồm các đỉnh của đồ thị.
- E là tập hợp các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung.

2.1.2 Một số khái niệm:

Định nghĩa 2.3: Cho đồ thị vô hướng G=(V,E).

- Hai đỉnh u và v của đồ thị được gọi là kề nhau nếu (u,v) là một cạnh của đồ thị.

- Nếu e = (u,v) là cạnh của đồ thị thì ta nói cạnh này là liên thuộc với hai đỉnh u và v. Cạnh được nói là nối đỉnh u và v. Đỉnh u và v được gọi là đỉnh đầu của cạnh e.

Định nghĩa 2.4: Cho đồ thị vô hướng G=(V,E). Bậc của đỉnh v trong đồ thị, ký hiệu là deg(v), là số cạnh liên thuộc với nó. Đỉnh có bậc 0 được gọi là đỉnh cô lập, đỉnh có bậc 1 gọi là đỉnh treo.

Định nghĩa 2.5: Cho đồ thị có hướng G=(V,E).

- Hai đỉnh u và v của đồ thị được gọi là kề nhau nếu (u,v) là một cung của đồ thị.
- Nếu e = (u,v) là cung của đồ thị thì ta nói cung này đi ra khỏi đỉnh u vào đi vào đỉnh v. Đỉnh u được gọi là đỉnh đầu của cung e và đỉnh v được gọi là đỉnh cuối của cung e.

Định nghĩa 2.6: Cho đồ thị có hướng G=(V,E)

- Nửa bậc ra của đỉnh v trong đồ thị, ký hiệu là $deg^+(v)$, là số cạnh đi ra khỏi v.
- Nửa bậc vào của đỉnh v trong đồ thị, ký hiệu là $deg^-(v)$, là số cạnh vào v.

2.2 Đường đi, chu trình, tính liên thông

Định nghĩa 2.7: Cho đồ thị G = (V,E). Đường đi độ dài n từ đỉnh u đến đỉnh v (n là số nguyên dương) là dãy:

$$x_0, x_1, \ldots, x_{n-1}, x_n$$

trong đó $u = x_0$, $v = x_n$, $(x_i x_{i+1}) \in E$, i = 0, 1, …, n-1.

Đường đi nói trên còn có thể được biểu diễn bằng dãy các cạnh/cung:

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$$

Đỉnh u gọi là đỉnh đầu của đường đi, đỉnh v gọi là đỉnh cuối của đường đi.

Đường đi có đỉnh đầu và đỉnh cuối trùng nhau (u=v) gọi là chu trình.

Định nghĩa 2.8: Đồ thị vô hướng G = (V,E) được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

Định nghĩa 2.9: Cho đồ thị G = (V,E). Đồ thị H = (W,F) được gọi là đồ thị con của G nếu và chỉ nếu $W \subseteq V$ và $F \subseteq E$.

Trong trường hợp một đồ thị vô hướng G không liên thông, nó sẽ được phân thành các đồ thị con độc lập nhau và chúng đều liên thông. Mỗi đồ thị con như vậy được gọi là một thành phần liên thông của G.

Định nghĩa 2.10. Cho G = (V,E) là đồ thị có hướng.

- G được gọi là liên thông mạnh nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.
- G được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó (đồ thị vô hướng có được bằng cách biến các cung một chiều thành các cạnh hai chiều) là đồ thị vô hướng liên thông.

2.3 Biểu diễn đồ thị trong máy tính

2.3.1 Ma trận kề, ma trận trọng số

Xét đồ thị đơn vô hướng G =(V, E), với tập đỉnh V = {1, 2, . . ., n}, tập cạnh E = {e1, e2, . . ., em}. Ta gọi ma trận kề của đồ thị G là ma trận có các phần tử hoặc bằng 0 hoặc bằng 1 theo qui định như sau:

$A = \{ a_{ij}: a_{ij} = 1$ nếu $(i, j) \in E$, $a_{ij} = 0$ nếu $(i,j) \notin E$; i, j =1, 2, . . ., n$\}$.

2.3.2 Danh sách cạnh (cung)

Mỗi cạnh (cung) e (x, y) được tương ứng với hai biến dau[e], cuoi[e]. Như vậy, để lưu trữ đồ thị, ta cần 2m đơn vị bộ nhớ. Nhược điểm lớn nhất của phương pháp này là để nhận biết những cạnh nào kề với cạnh nào chúng ta cần m phép so sánh trong khi duyệt qua tất cả m cạnh (cung) của đồ thị. Nếu là đồ thị có trọng số, ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

2.3.3 Danh sách kề

Trong biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó mà ta ký hiệu là Ke(v), nghĩa là

$$Ke(v) = \{ u \in V: (u, v) \in E\},$$

Với cách biểu diễn này, mỗi đỉnh i của đồ thị, ta làm tương ứng với

một danh sách tất cả các đỉnh kề với nó và được ký hiệu là List(i). Để biểu diễn List(i), ta có thể dùng các kiểu dữ liệu kiểu tập hợp, mảng hoặc danh sách liên kết.

2.4 Các thuật toán tìm kiếm trên đồ thị

2.4.1 Thuật toán tìm kiếm theo chiều sâu

Thuật toán tìm kiếm theo chiều sâu bắt đầu từ đỉnh v nào đó sẽ duyệt tất cả các đỉnh liên thông với v. Thuật toán có thể được mô tả bằng thủ tục đệ qui DFS () trong đó: chuaxet - là mảng các giá trị logic được thiết lập giá trị TRUE

```
procedure DFS( v);
begin
    Thăm_Đỉnh(v); chuaxet[v] := FALSE;
    for  u ∈ ke(v)  to n do
    begin
            if (chuaxet[u] ) then
                    DFS(A, n, v, chuaxet);
end;
```

2.4.2 Thuật toán tìm kiếm theo chiều rộng (Breadth First Search)

Khác với thuật toán tìm kiếm theo chiều sâu, thuật toán tìm kiếm theo chiều rộng thay thế việc sử dụng stack bằng hàng đợi queue. Trong thủ tục này, đỉnh được nạp vào hàng đợi đầu tiên là v, các đỉnh kề với v (v_1, v_2, . . ., v_k) được nạp vào queue kế tiếp. Quá trình được thực tương tự với các đỉnh trong hàng đợi. Thuật toán dừng khi ta đã duyệt hết các đỉnh kề với đỉnh trong hàng đợi.

```
chuaxet- mảng kiểm tra các đỉnh đã xét hay chưa;
queue – hàng đợi lưu trữ các đỉnh sẽ được duyệt của đồ thị;
procedure BFS(u);
begin
    queue := ϕ;
    u <= queue; (*nạp u vào hàng đợi*)
```

```
        chuaxet[u] := false;
        while (queue ≠ ϕ ) do
        begin
                queue<=p; (* lấy p ra từ stack*)
                Thăm_Đỉnh(p);
                for  v ∈ ke(p) do
                        if (chuaxet[v] ) then
                        begin
                                v<= queue; (*nạp v vào hàng đợi*)
                                chuaxet[v] := false;
                        end;
        end;
    end;
```

2.4.3 Kiểm tra tính liên thông của đồ thị

Bài toán: Cho đồ thị G=(V, E). Trong đó V là tập đỉnh, E là tập cạnh của đồ thị. Hãy tìm số thành phần liên thông của đồ thị và cho biết mỗi thành phần liên thông của đồ thị gồm những đỉnh nào?

Nếu đồ thị là liên thông, ta chỉ cần gọi tới thủ tục DFS() hoặc BFS() một lần. Nếu đồ thị là không liên thông, khi đó số thành phần liên thông của đồ thị chính bằng số lần gọi tới thủ tục BFS() hoặc DFS().

2.4.4 Tìm đường đi giữa hai đỉnh bất kỳ của đồ thị

Bài toán: Cho đồ thị G=(V, E). Trong đó V là tập đỉnh, E là tập cạnh của đồ thị. Hãy tìm đường đi từ đỉnh s∈ V tới đỉnh t∈ V của G.

Thủ tục BFS(s) hoặc DFS(s) cho phép ta duyệt các đỉnh cùng một thành phần liên thông với đỉnh s. Như vậy, nếu trong số các đỉnh liên thông với s chứa t thì chắc chắn có đường đi từ đỉnh s đến đỉnh t. Nếu trong số các đỉnh liên thông với đỉnh s không chứa đỉnh t thì không tồn tại đường đi từ đỉnh s đến đỉnh t. Do vậy, chúng ta chỉ cần gọi tới thủ tục DFS(s) hoặc BFS(s) và kiểm tra xem đỉnh t có thuộc thành phần liên thông với s hay không.

2.5 Đồ thị Euler và Hamilton

2.5.1 Đồ thị Euler

Định nghĩa 2.11: Cho đồ thị G = (V,E). Chu trình đơn trong G đi qua tất cả các cạnh của nó được gọi là chu trình Euler. Đường đi đơn đi qua tất cả các cạnh của đồ thị được gọi là đường đi Euler.

Đồ thị G được gọi là đồ thị Euler nếu nó có chứa chu trình Euler. G được gọi là đồ thị nửa Euler nếu nó có chứa đường đi Euler.

2.5.2 Đồ thị Hamilton

Định nghĩa 2.12: Cho đồ thị G = (V,E). Chu trình sơ cấp trong G đi qua tất cả các đỉnh của nó được gọi là chu trình Hamilton. Đường đi sơ cấp đi qua tất cả các đỉnh của đồ thị được gọi là đường đi Hamilton.

Đồ thị G được gọi là đồ thị Hamilton nếu nó có chứa chu trình Hamilton. G được gọi là đồ thị nửa Hamilton nếu nó có chứa đường đi Hamilton.

Đồ thị Hamilton cũng là đồ thị nửa Hamilton, ngược lại thì không phải lúc nào cũng đúng.

2.6 Cây và cây bao trùm của đồ thị

2.6.1 Cây và các tính chất cơ bản của cây

Định nghĩa 2.13: Cây là một đồ thị vô hướng liên thông và không chứa chu trình.

+ *Gốc* của cây là một đỉnh đặc biệt, thông thường là đỉnh trên cùng.

+ *Mức* của đỉnh là độ dài đường đi từ gốc đến đỉnh đó.

+ *Độ cao* của cây là mức lớn nhất của cây (tức mức của đỉnh cách xa cây nhất).

2.6.2 Cây bao trùm của đồ thị

Định nghĩa 2.14: Cho đồ thị G=(V,E). Cây T gọi là cây bao trùm của G, nếu T là đồ thị con phủ của G.

Định lý 2.8: Đồ thị G=(V,E) có cây bao trùm khi và chỉ khi G liên thông.

2.6.3 Cây bao trùm nhỏ nhất

Bài toán tìm cây bao trùm nhỏ nhất là một trong những bài toán tối ưu trên đồ thị có ứng dụng trong nhiều lĩnh vực khác nhau của thực tế. Bài toán được phát biểu như sau:

Cho đồ thị vô hướng G = (V,E,w). Hãy tìm cây bao trùm T của G sao cho tổng trọng số của các cạnh của T đạt giá trị nhỏ nhất.

Để tìm một cây bao trùm chúng ta có thể thực hiện theo các bước như sau:

Bước 1. Thiết lập tập cạnh của cây bao trùm là ϕ . Chọn cạnh e = (i, j) có độ dài nhỏ nhất bổ sung vào T.

Bước 2. Trong số các cạnh thuộc E \ T, tìm cạnh e = (i_1, j_1) có độ dài nhỏ nhất sao cho khi bổ sung cạnh đó vào T không tạo nên chu trình. Để thực hiện điều này, chúng ta phải chọn cạnh có độ dài nhỏ nhất sao cho hoặc $i_1 \in$ T và $j_1 \notin$ T, hoặc $j_1 \in$ T và $i_1 \notin$ T.

Bước 3. Kiểm tra xem T đã đủ n-1 cạnh hay chưa? Nếu T đủ n-1 cạnh thì nó chính là cây bao trùm ngắn nhất cần tìm. Nếu chưa đủ n-1 cạnh thì thực hiện lại bước 2.

CHƯƠNG 3
XÂY DỰNG THUẬT TOÁN SONG SONG CHO MỘT SỐ BÀI TOÁN VỀ ĐỒ THỊ

3.1 Bài toán tìm cây bao trùm nhỏ nhất

3.1.1 Bài toán:

Cho đồ thị vô hướng có trọng số G =(V, E, w). Hãy tìm cây bao trùm T của G sao cho tổng trọng số của các cạnh của T đạt giá trị nhỏ nhất.

3.1.2 Thuật toán Prim tìm cây bao trùm nhất

Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất. Trong phương pháp này, bắt đầu từ một đỉnh tùy ý s của đồ thị, đầu tiên, ta nối s với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh t. Kế tiếp, trong số các cạnh

nối với s hay t, ta lại chọn cạnh có trọng số nhỏ nhất, … cho đến khi ta thu được cây gồm n đỉnh và n-1 cạnh. Đây chính là cây khung nhỏ nhất cần tìm.

+ Thuật toán Prim được mô tả như sau : Để biểu diễn lời giải, ta sẽ sử dụng 2 mảng:

- Mảng d: d[v] dùng để lưu độ dài cạnh ngắn nhất nối với v trong số các cạnh chưa xét.

- Mảng near: near[v] dùng để lưu đỉnh còn lại (ngoài v) của cạnh ngắn nhất nói ở trên.

+ Đầu vào: Đồ thị G = (V,E,w)

+ Đầu ra: Cây bao trùm nhỏ nhất của G và trọng số tương ứng

```
1. Procedure Prim(V, E, w)
2. Begin (* Khởi tạo *)
3.      Chọn s là một đỉnh nào đó của đồ thị.
4.      H := {s}; (* Tập những đỉnh đã đưa vào cây *)
5.      T := ∅; (* Tập cạnh của cây *)
6.      d[s] = 0;
7.      near[s] = s;
8.      For  v∈(V \ H)  do
9.              Begin
10.                     d[v] := a[s,v];
11.                     near[v] := s;
12.             End;
13.     Stop := False;
14.     While (not Stop) do (* Bước lặp *)
15.             Begin
16              Tìm u∈(V \ H) thỏa mãn d[u] = min{d[v]: v∈(V \ H)};
17.                     H := H ∪ {u};
18.                     T := T ∪ { (u, near[u]) };
```

```
19.                 If  |H| = n then
20.                     Begin
21.                         C := (H, T) là cây khung của đồ thị.
22.                         Stop := True;
23.                     End;
24.                 Else
25.                 For v ∈ (V \ H) do
26.                     If d[v] > a[u,v] then
27.                         Begin
28.                             d[v] := a[u,v];
29.                             near[v] := u;
30.                         End;
31.             End;
32.     End;
```

+ Độ phức tạp của thuật toán Prim là Θ(n2). Thuật toán sử dụng 2 vòng lặp, mỗi vòng có (n-1) lần lặp.

$$T(n)=2(n-1)(n-1) \in O(n^2)$$

Do đó độ phức tạp của thuật toán là $O(n^2)$

3.1.3 Xây dựng thuật toán song song

+ Chia dữ liệu cho thuật toán Prim:

- Giả thiết đồ thị có n đỉnh, thuật toán thực hiện trên p BXL.
- Tập các đỉnh V của đồ thị được chia thành p tập con, mỗi tập con gồm n/p đỉnh liền kề và mỗi tập được gán để thực hiện trên một BXL.
- BXL P_i quản lí một tập con V_i là các cột liên tiếp của ma trận kề và số dòng là n (tương ứng với số đỉnh của đồ thị).
- Tại bước kết nạp đỉnh u vào H, BXL P_i sẽ tính $d_i[u]=\min\{d[v]:v\in(V_i \setminus H)\}$;
- Trọng số của cây bao trùm nhỏ nhất của đồ thị thu được từ các $d_i[u]$ bằng cách tổng hợp kết quả từ các BXL, lấy d_i min[u] và chuyển về P_0.

- BXL P_0 nhận đỉnh u mới tìm được và chèn vào H (H lưu các đỉnh đã được đưa vào cây) và phát đỉnh u đến tất cả các BXL khác.

+ Thuật toán song song:

Bước 1: Khởi tạo

H := {s}; T := ∅;

d[s] = 0; d[v] := a[s,v];

Chia ma trận kề đến các BXL, mỗi BXL P_i quản lí một tập đỉnh V_i

Bước 2:

BXL P_0 phát tán đỉnh được chọn là s đến các BXL còn lại.

Bước 3: Lặp lại cho đến khi có n đỉnh được đưa vào H (với n là số đỉnh của đồ thị).

BXL Pi tính $d_i[u] = \min\{d[v]: v \in (V_i \setminus H)\}$; gửi $d_i[u]$ về P_0

BXL P_0 chọn d_i min [u] từ các $d_i[u]$

BXL P_0 đưa đỉnh được chọn vào tập H và phát tán đỉnh được chọn là u đến các BXL còn lại.

+ Độ phức tạp của thuật toán:

Khi một đỉnh u mới được thêm vào T, các giá trị của d [v] với $v \in (V \setminus H)$ được cập nhật. BXL đảm nhiệm đỉnh v tính trọng số của các cạnh (u, v). Do đó, mỗi BXL P_i cần để lưu trữ các cột của ma trận kề tương ứng của đỉnh V_i đỉnh được giao. Vì vậy độ phức tạp của mỗi BXL sẽ là $O(n^2/p)$.

Các BXL tìm cạnh có trọng số nhỏ nhất và cập nhật các giá trị d[v] trong mỗi lần lặp là O(n/p). Trong một BXL thời gian thực hiện tìm giá trị nhỏ nhất trong mỗi lần lặp là O(logp). Thời gian thực hiện thuật toán song song là:

$$T_P = O\left\{\frac{n^2}{p}\right\} + O(n \log p)$$

So với thuật toán tuần tự độ phức tạp của thuật toán song song cũng tương đương nhau. Kết quả chỉ thể hiện được qua thực nghiệm chương

trình.

3.2 Bài toán tìm đường đi ngắn nhất xuất phát từ một đỉnh:

3.2.1 Bài toán:

Cho đồ thị có trọng số G = (V,E,w) mỗi cạnh của đồ thị được gán một trọng số (giá trị). Tìm đường đi ngắn nhất từ một đỉnh xuất phát $s \in V$ (đỉnh nguồn) đến đỉnh cuối $v \in V$ (đỉnh đích).

3.2.2 Thuật toán Dijkstra:

Thuật toán Dijkstra thực hiện việc gán và giảm giá trị của nhãn tại mỗi đỉnh i của đồ thị G.

Để đơn giản việc tính toán, ta xây dựng ma trận trọng số a như sau:

$$a[i,j] = \begin{cases} w(i,j) \text{ , nếu } (i, j) \in E \textit{ (trọng số cạnh (i,j))} \\ \infty \text{ , nếu } (i, j) \notin E \\ 0 \text{ , nếu } i = j. \end{cases}$$

Khi đó, thuật toán Dijkstra được trình bày như sau :

- Đầu vào: Đồ thị lên thông có trọng số G=(V,E,w) với n đỉnh, $s \in V$ là đỉnh xuất phát. Giả thiết : $a[u,v] \geq 0$
- Đầu ra: khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại của G

+ Một số kí hiệu:

- s: đỉnh xuất phát (nguồn)
- v: đỉnh kết thúc (đích)
- truoc[v], $v \in V$ ghi lại đỉnh trước v trong đường đi ngắn nhất từ s đến v.
- a[i,j]: trọng số của cạnh (i,j)
- d[v]: khoảng cách ngắn nhất từ s đến v.
- T là tập các đỉnh có nhãn tạm thời

```
1. Procedure Dijkstra(G,V,w, s);
2. Begin // Khởi tạo nhãn tạm thời cho các đỉnh
3. For  v∈ V do
4. Begin
```

```
5.      d[v]:=a[s, v];
6.      Truoc [v]:=s; // Truoc[v] ghi lại đỉnh trước v trong đường
đi ngắn nhất từ s đến v.
7.end;
8.      d[s]:=0;
9.      T:=V\{s}; // T là tập các đỉnh có nhãn tạm thời
10. While T≠ ∅ do // Bước lặp
11. Begin
12.     Tìm đỉnh u∈ T thỏa mãn d[u]=min {d[v]:v∈ T};
13.     T:=T\{u};// cố định nhãn của đỉnh u
14.         For v∈ T do // gán nhãn lại cho các đỉnh trong T
15.             If d[v]>d[u]+a[u,v] then
16.             Begin
17.                 d[v]:=d[u]+a[u,v];
28.                 truoc[v]:=u;
29.             end;
20.     end;
21. end;
```

+ Đánh giá số phép toán cần thực hiện của thuật toán: Ở mỗi bước lặp để tìm ra điểm u cần thực hiện O(n) phép toán, để gán nhãn lại cũng cần thực hiện một số lượng phép toán cũng là O(n). Thuật toán cần phải thực hiện n-1 bước lặp, vậy thời gian tính toán của thuật toán là $O(n^2)$.

3.2.3 Xây dựng thuật toán song song

+ Chia dữ liệu cho thuật toán Dijkstra song song:

- Giả thiết đồ thị có n đỉnh, thuật toán song song thực hiện trên p BXL .
- Tập các đỉnh V của đồ thị được chia thành p tập con, mỗi tập con gồm n/p đỉnh liền kề và được gán để tính trên một BXL.
- BXL P_i quản lí một tập con V_i là các cột liên tiếp của ma trận kề và số dòng là n (tương ứng số đỉnh của đồ thị).

- Tại bước gán nhãn cho đỉnh u, BXL P_i sẽ tính $d_i[u]=\min\{d[v]:v\in V_i\}$;
- Trọng số đường đi ngắn nhất của đồ thị thu được trên tất cả các $d_i[u]$ bằng cách tổng hợp kết quả từ các BXL và được chuyển về P_0.
- BXL P_0 nhận đỉnh u mới tìm được và chèn vào T (T lưu các đỉnh đã được gán nhãn) và phát đỉnh u đến các BXL khác.

+ Thuật toán song song được mô tả như sau:

Bước 1: Khởi tạo

T := V \ {s}; d[s] = 0;

d[v] := a[s,v];

truoc[v] := s;

Chia ma trận kề đến các BXL, mỗi BXL Pi quản lí một tập đỉnh V_i

Bước 2:

BXL P_0 phát tán đỉnh được chọn là s đến các BXL khác.

Bước 3: Lặp lại cho đến khi đỉnh đích được chọn.

BXL P_i tính $d_i[u]=\min\{d[v]:v\in V_i\}$; chọn đỉnh có nhãn nhỏ nhất từ các $d_i[u]$ gửi về BXL P_0

BXL P_0 phát tán đỉnh được chọn là u đến các BXL còn lại.

BXL P_0 đưa đỉnh được chọn vào tập T.

+ Độ phức tạp của thuật toán:

Thuật toán Dijkstra song song sử dụng n BXL, mỗi BXL quản lí n/p đỉnh của đồ thị. Mỗi BXL P_i tìm đường đi ngắn nhất từ đỉnh v đến tất cả các đỉnh khác bằng cách thực hiện thuật toán Dijkstra tuần tự một đỉnh. Vì vậy, thời gian thực hiện thuật toán song song là:

$$T_p = O\left\{\frac{n^2}{p}\right\} + O(n\log p)$$

So với thuật toán tuần tự độ phức tạp của thuật toán song song cũng tương đương nhau. Kết quả chỉ thể hiện được qua thực nghiệm chương trình.

4. KẾT QUẢ THỬ NGHIỆM:

Chương trình demo thử nghiệm thuật toán song song viết bằng ngôn ngữ

C++ sử dụng thư viện omp.h trên visual C++ 2008, sử dụng máy tính thử nghiệm có cấu hình như sau:

Bộ xử lí	Intel Core i3-370, 2.4 GHz
Bộ nhớ	2 GB

+ Kết quả thử nghiệm:

Số đỉnh	**Prim tuần tự**	**Prim song song**	**Dijkstra tuần tự**	**Dijkstra song song**
8	3 ms	3 ms	2 ms	2 ms
32	15 ms	12 ms	8 ms	6 ms
64	41 ms	31 ms	17 ms	11 ms
128	63 ms	47 ms	31 ms	18 ms
256	91 ms	63 ms	45 ms	24 ms

Qua kết quả thử nghiệm chương trình có thể nhận thấy rằng với đồ thị có số đỉnh càng nhiều thì kết quả thực hiện của chương trình theo mô hình song song càng tốt hơn so với chương trình theo mô hình tuần tự.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Luận văn đã nghiên cứu tổng quát về thuật toán song song và áp dụng cho một số bài toán về đồ thị. Luận văn cũng đã khái quát các khái niệm về thuật toán song song và các vấn đề liên quan đến thuật toán song song, các mô hình song song, môi trường lập trình song song, lý thuyết đồ thị, cách biểu diễn đồ thị trên máy tính, các vấn đề về đường đi, cây bao trùm trên đồ thị.

Từ các hiểu biết trên luận văn đã nghiên cứu để song song hóa các thuật toán tìm đường đi ngắn nhất (Dijkstra) và cây bao trùm nhỏ nhất (Prim) từ các thuật toán tuần tự truyền thống.

Ngoài ra do còn thiếu kinh nghiệm trong việc phân tích, thiết kế và cài đặt thuật toán song song nên kết quả đạt được còn hạn chế.

Qua tìm hiểu, nghiên cứu đề tài này cũng đã nắm được các kiến thức về xử lí song song, lập trình song song, lý thuyết đồ thị. Tìm hiểu được cách xây dựng thuật toán song song và đã áp dụng vào một số bài toán cụ thể.

Với kết quả đạt được, tác giả mong muốn có các nghiên cứu thêm để phát triển đề tài này để cải tiến thêm các thuật toán và có thể áp dụng vào các lĩnh vực khác như:

1. Áp dụng song song hóa các thuật toán Dijkstra, Prim theo mô hình máy tính phân cụm.
2. Nghiên cứu mô hình lập trình song song và áp dụng chuyển các thuật toán tuần tự khác về đồ thị sang thuật toán song song.
3. Ứng dụng thực tế vào các bài toán cụ thể trong nhiều lĩnh vực khác.