NÂNG CAO HIỆU NĂNG TÍNH TOÁN CHO THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ MỞ RỘNG

IMPROVING COMPUTING PERFORMANCE FOR ALGORITHM IN FINDING THE SHORTEST PATH IN EXTENDED GRAPH

Nguyễn Đình Lầu[1], Trần Quốc Chiến[2], Trần Ngọc Việt[1]

[1]Trường Cao đẳng Giao thông Vận tải II, Email: trviet01@yahoo.com, launhi@gmail.com

[2]Trường Đại học Sư phạm, Đại học Đà Nẵng; Email: tqchien@dce.udn.vn

Tóm tắt - Đồ thị là công cụ toán học hữu ích ứng dụng trong nhiều lĩnh vực như giao thông, công nghệ thông tin, kinh tế,…Thuật toán tìm đường đi ngắn nhất trên đồ thị mở rộng đã được công bố trong [1]. Trong bài báo này, chúng tôi trình bày chi tiết thuật toán tuần tự tìm đường đi ngắn nhất giữa hai đỉnh trên đồ thị mở rộng và chúng tôi xây dựng thuật toán này trên đa bộ xử lý để nâng cao hiệu năng tính toán. Các định lý và mệnh đề trong bài báo được chứng minh, phần thực nghiệm cho kết quả chính xác. Thuật toán song song tìm đường đi giữa hai đỉnh trên đồ thị mở rộng được xây dựng trên k bộ xử lý. Hệ thống thực nghiệm ở đây là mạng LAN và chương trình được xây dựng bằng ngôn ngữ Java.

Từ khóa - song song; đồ thị; mở rộng; thuật toán; đường đi ngắn nhất.

Abstract - The graph is a powerful mathematical tool applied in many fields such as transportation, communication, information technology, economy,… Algorithm finding the shortest path in extended graph was proposed in [1]. In this paper we present and demonstrate in details the sequential algorithm to find the shortest path between two vertices on the extended graph and build this algorithm on multiple processors to improve computing performance. The properties and theorems of this paper are carefully proven and the experiment shows correct results. Parallel algorithm finding the shortest path between the two vertices in the extended graph is built on k processors. The experimental system used is LAN network and the program written is in Java.

Key words - parallel; graph; extended; algorithm; the shortest path.

1. Đặt vấn đề

Cho đến nay, trong đồ thị chỉ xét đến trọng số của các cạnh, các đỉnh một cách độc lập, trong đó độ dài đường đi chỉ đơn thuần là tổng trọng số các cạnh và các đỉnh trên đường đi đó. Tuy nhiên, trong nhiều bài toán thực tế, trọng số tại một đỉnh không giống nhau với mọi đường đi qua đỉnh đó mà còn phụ thuộc vào cạnh đi đến và cạnh đi khỏi đỉnh đó [1].

Ví dụ thời gian đi qua ngã tư trên mạng giao thông phụ thuộc vào hướng di chuyển của phương tiện giao thông: rẽ phải, đi thẳng hay rẻ trái. Do đó cần xây dựng một mô hình đồ thị mở rộng để có thể áp dụng mô hình hóa các bài toán thực tế chính xác và hiệu quả hơn. Thuật toán tìm đường đi ngắn nhất là thuật toán cơ sở được sử dụng trong nhiều bài toán tối ưu trên đồ thị và mạng như: ứng dụng để xây dựng bài toán phân luồng tối ưu [4] và ứng dụng để xây dựng thuật toán tìm luồng cực đại chi phí giới hạn [5], đồng thời thuật toán này cũng được ứng dụng để phân luồng giao thông ở Thành phố Đà Nẵng (đề tài cấp Thành phố Đà Nẵng) của chính chúng tôi.

Vì vậy, trong bài báo này, mô hình đồ thị mở rộng được định nghĩa và giới thiệu thuật toán tuần tự tìm đường đi ngắn nhất giữa hai đỉnh trên đồ thị mở rộng được kế thừa từ công trình [1]. Hơn nữa, cho đến nay các công trình trên thế giới [7, 8, 9, 10] chỉ thực hiện song song trên đồ thị cổ điển, tức là đồ thị không có chi phí đỉnh. Để cho bài toán tìm đường đi ngắn nhất trên mạng đồ thị mở rộng xử lý nhanh về thời gian, chúng tôi xây dựng theo hướng song song để nâng cao hiệu năng tính toán của thuật toán này.

Kỹ thuật để xây dựng thuật toán song song là dùng k bộ xử lý. Trong k bộ xử lý đó ta chọn 1 bộ xử lý đóng vai trò là bộ xử chính (Server), k-1 bộ xử lý còn lại đóng vai trò là bộ xử lý phụ (Client). Server sẽ quản lý dữ liệu và chia dữ liệu cho các Client. Các Client tìm giá trị nhãn nhỏ nhất trên các đỉnh mà mình nắm giữ, sau đó gửi về Server. Server sẽ tính min của các giá trị mà các bộ xử lý phụ gửi đến, sau đó gửi nhãn đỉnh-cạnh tương ứng với giá trị min này lên cho các Client để tiếp tục tính toán cho đến khi tìm được kết quả và kết thúc. Hệ thống thực nghiệm ở đây là mạng LAN và chương trình được xây dựng bằng ngôn ngữ Java với hệ quản trị cơ sở dữ liệu MySQL.

2. Giới thiêu đồ thị mở rộng

Đồ thị mở rộng đã được công bố trong [1]. Trong phần này, chúng tôi chỉ giới lại đồ thị mở rộng được kế thừa từ [1].

Cho đồ thị hỗn hợp G(V, E) với tập đỉnh V và tập cạnh E, trong đó các cạnh có thể có hướng hoặc vô hướng. Mỗi cạnh e∈ E được gán trọng số $w_E(e)$. Với mỗi đỉnh v ∈ V, ký hiệu E_v là tập các cạnh liên thuộc đỉnh v. Mỗi đỉnh v ∈ V và mỗi cạnh (e,e') ∈ E_vx E_v, e≠e' được gán trọng số w_V(v,e,e').

Bộ (V, E, w_E, w_V) gọi là đồ thị mở rộng. Cho p là đường đi từ u đếnv qua các cạnh e_i, i=1,…,h+1, và các đỉnh u_i, i=1,…,h, như sau: P=[u, e_1, u_1, e_2, u_2, …, e_h, u_h, e_{h+1}, v]

Định nghĩa độ dài đường đi p, ký hiệu l(p), theo công thức sau:

$$l(p) = \sum_{i=1}^{h+1} w_E(e_i) + \sum_{i=1}^{h} w_V(u_i, e_i, e_{i+1}) \quad (1)$$

Bài toán tìm đường đi ngắn nhất:

Cho đồ thị mở rộng G=(V, E, w_E, w_V) và các đỉnh s, t∈V. Tìm đường đi ngắn nhất từ s đến t

3. Thuật toán tìm đường đi ngắn nhất giữa 2 đỉnh trong đồ thị mở rộng

Thuật toán này đã được công bố trong [1], tuy nhiên để thuật toán song song kế thừa các ký hiệu, khái niệm các biến từ thuật toán tuần tự, cũng như để đơn giản hơn cho

việc xây dựng thuật toán song song chúng tôi giới thiệu cụ thể lại thuật toán này từ [1].

- *Đầu vào*: Đồ thị mở rộng G(V, E, w_e, w_v), và các đỉnh s, t∈V.

- *Đầu ra:* l(t) là chiều dài đường đi ngắn nhất từ s đến t và đường đi ngắn nhất (nếu l(t) <+∞).

- *Phương pháp:* Thuật toán sử dụng các ký hiệu sau:

S là tập đỉnh đã tìm được đường đi ngắn nhất xuất phát từ s. T=V-S;

l(v) là độ dài đường đi ngắn nhất từ s đến v.

VE={(v,e)|v∈V\{s}& e∈E_V} ∪ {(s, ϕ)} là tập các đỉnh và cạnh liên thuộc.

SE là tập các đỉnh-cạnh bị loại khỏi VE, TE=VE-SE, L(v,e) là nhãn cặp đỉnh- cạnh (v,e) ∈ VE

P(v,e) là cặp đỉnh- cạnh trước (v,e) ∈ VE

Bước 1. (Khởi tạo)

Đặt S= ϕ ; T=V; VE={(v,e)|v∈V\{s}& e∈E_V} ∪ {(s, ϕ)} SE= ϕ ; TE=VE;

Gán L(v,e)=∞, ∀ (v,e) ∈ VE, L(s, ϕ):=0. Gán P(v,e)= ϕ ∀ (v,e) ∈VE

Bước 2. Tính m = min{L((v,e))| (v,e) ∈ TE}.

Nếu m=+∞, kết luận; không tồn tại đường đi từ s đến t. *Kết thúc*.

Ngược lại, nếu m<+∞, chọn (v_{min}, e_{min}) ∈TE sao cho L(v_{min}, e_{min})=m, đặt TE=TE-{(v_{min}, e_{min})}, SE=SE ∪ {(v_{min}, e_{min})}, sang bước 3.

Bước 3. Nếu $v_{min} \notin S$, thì đặt le(v_{min})=e_{min}

S=S ∪ $\{v_{min}\}$, l(v_{min})=L(v_{min}, e_{min}),

T=T-{ v_{min}},

Nếu t∈ v_{min}, sang bước 5, ngược lại sang bước 4.

Bước 4. Với mỗi (v,e) ∈ TE kề (kề sau) (v_{min},e_{min}),

đặt L'(v,e)=L(v_{min},e_{min})+w_E(v_{min},v)+w_V(v_{min},e_{min},e) nếu v_{min} ≠ s và L'(v,e)=L(s, ϕ)+ w_E(v_{min},v) nếu v_{min} = s

Nếu L(v,e)>L'(v,e), thì gán L(v,e)=L'(v,e) và P(v,e)= (v_{min}, e_{min})

Quay về bước 2.

Bước 5. (Tìm đường đi ngắn nhất)

Án l(t)=L(t,le(t)) là chiều dài đường đi ngắn nhất từ s đến t. Từ t lần ngược theo đỉnh-cạnh trước ta nhận được đường đi ngắn nhất như sau: đặt (v_1,e_1)=P(t,le(t)), (v_2,e_2)=P(v_1,e_1), …, (v_k,e_k) =P(v_{k-1},e_{k-1}), (s, ϕ)= P(v_k,e_k).

Suy ra đường đi ngắn nhất là:

s→v_k→v_{k-1}→…→v_1→t. Kết thúc.

Nhận xét 1:

Thuật toán tìm đường đi ngắn nhất giữa hai đỉnh trong đồ thị mở rộng là đúng [1].

Nhận xét 2:

Cho G là đồ thị mở rộng có n đỉnh. Khi đó độ phức tạp của thuật toán là O(n^3) [1].

4. Thuật toán song song tìm đường đi ngắn nhất giữa 2 đỉnh trong đồ thị mở rộng

- Ý tưởng của thuật toán.

Chúng tôi xây dựng thuật toán song song trên k bộ xử lý (P_0, P_1,…,P_{k-1}) [2, 3, 6]. Trong k bộ xử lý đó có một bộ xử lý chính (P_0) quản lý dữ liệu, chia dữ liệu cho k-1 bộ xử lý phụ (P_1,…,P_{k-1}). Các bộ xử lý phụ nhận dữ liệu và tìm L(v,e) nhỏ nhất trên các đỉnh mà mình nắm giữ và gửi về bộ xử lý chính. Bộ xử lý chính sẽ tìm L(v_{min}, e_{min}) =min(L_i(v,e)), i=0,…,k-1) của các bộ xử lý phụ gửi đến. Sau đó bộ xử lý chính sẽ gửi (v_{min}, e_{min}) đến các bộ xử lý phụ để các bộ xử lý tiếp tục tính toán.

- *Đầu vào*: Đồ thị mở rộng G(V, E, w_E, w_V), đỉnh s, t∈V, k bộ xử lý (P_0, P_1,…,P_{k-1}).

- *Đầu ra:* l(t) là chiều dài đường đi ngắn nhất từ s đến t và đường đi ngắn nhất (nếu l(t) <+∞).

- *Phương pháp:*

Bước 1. Bộ xử lý chính thực hiện

Chia trọng số cạnh A(w_E), số đỉnh và tập các cặp đỉnh và cạnh liên thuộc VE của đồ thị cho k bộ xử lý

Xây dựng T_i (i=0,…,k-1) là tập đỉnh mà bộ xử lý P_i (i=0,…,k-1) nhận

Xây dựng VE_i (i=0,…,k-1) là tập các cặp đỉnh và cạnh liên thuộc mà bộ xử lý P_i (i=0,…,k-1) nhận

Xây dựng A_i (i=0,…,k-1) là ma trận trong số cạnh mà bộ xử lý P_i (i=0,…,k-1) nhận

Khởi tạo tập T=V, S= ϕ là các đỉnh đã tìm được đường đi ngắn nhất xuất phát từ s

SE là tập các đỉnh-cạnh bị loại khỏi VE, L(v,e) là nhãn cặp đỉnh- cạnh (v,e) ∈ VE

P(v,e) là cặp đỉnh- cạnh trước (v,e)∈VE. Gửi T_i, A_i, VE_i (i=1,..,k-1) và trong số w_V đến k-1 bộ xử lý phụ

Bước 2. k bộ xử lý thực hiện.

Nhận dữ liệu từ bước 1. Bộ xử lý chính P_0 gán L(s, ϕ):=0;

Trong k bộ xử lý khởi tạo. TE_i=VE_i (i=0,…,k-1), SE_i= ϕ ,(i=0,…,k-1)

Trong k bộ xử lý, gán L(v,e)=∞ và P(v,e)= ϕ ∀ (v,e) ∈ VE_i (i=0,…,k-1)

Bước 3. k bộ xử lý thực hiện

Trên k bộ xử lý tìm m_i = min{L($(v,e)_j$)| $(v,e)_j$ ∈ TE_j (j=0,…,k-1) }. Các bộ xử lý phụ gửi m_i (i=1,…,k-1) và (v,e) tương ứng vừa tìm được đến bộ xử lý chính

Bước 4. Bộ xử lý chính thực hiện

Bộ xử lý chính tìm m=min{m_i, i=0,…,k-1} =L(v,e), nếu m=∞, sang bước 7.

Ngược lại, nếu m<∞ chuyển m lên các bộ xử lý phụ.

Bước 5. k bộ xử lý phụ thực hiện

k bộ xử lý Chọn (v_{min}, e_{min}) ∈ TE_i (i=0,…,k1) | L(v_{min}, e_{min})=m.

đặtTE_i=TE_i-{(v_{min}, e_{min})}.

SE_i= SE_i ∪ {(v_{min}, e_{min})}.

Gửi v_{min}, $L(v_{min}, e_{min})$ lên bộ xử lý chính

Bước 6. Bộ xử lý chính thực hiện

Nếu $v_{min} \notin S$, thì đặt $le(v_{min})=e_{min}$

$S=S \cup \{v_{min}\}$, $l(v_{min})=L(v_{min}, e_{min})$,

$T=T-\{v_{min}\}$

Nếu $t= v_{min}$, sang bước 8, ngược lại sang bước 7

Bước 7. *k bộ xử lý thực hiện*

Với mỗi $(v,e) \in TE_i$ kề (kề sau) (v_{min}, e_{min}), đặt

$L'(v,e)=L(v_{min}, e_{min})+w_E(v_{min},v)+w_V(v_{min},e_{min},e)$ nếu $v_{min} \neq s$ và $L'(v,e)=L(s, \phi)+ w_E(v_{min},v)$ nếu $v_{min} = s$

Nếu $L(v,e)>L'(v,e)$, thì gán $L(v,e)=L'(v,e)$ và $P(v,e)=(v_{min}, e_{min})$

Quay về bước 3.

Bước 8. k-1 bộ xử lý phụ gửi kết quả về bộ xử lý chính và kết thúc.

Bộ xử lý chính nhận kết quả từ các bộ xử lý phụ và tìm đường đi ngắn nhất như sau:

Gán $l(t)=L(t,le(t))$ là chiều dài đường đi ngắn nhất từ s đến t. Từ t lần ngược theo đỉnh-cạnh trước ta nhận được đường đi ngắn nhất như sau:

Đặt $(v_1,e_1)=P(t,le(t))$, $(v_2,e_2)=P(v_1,e_1)$, …, $(v_k,e_k)=P(v_{k-1},e_{k-1})$, $(s, \phi)= P(v_k,e_k)$.

Suy ra đường đi ngắn nhất là

$s \rightarrow v_k \rightarrow v_{k-1} \rightarrow \ldots \rightarrow v_1 \rightarrow t$. Kết thúc.

Định lý: Độ phức tạp của thuật toán song song là $O(\frac{n^3}{k}) + O(n \log k)$

Chứng minh:

Theo nhận xét 2 ở trên thì độ phức tạp của thuật toán tuần tự là $O(n^3)$.

Số đỉnh của mỗi bộ xử lý là n/k

Vì số cạnh liên thuộc mỗi đỉnh nhiều nhất là (n-1), cho nên tập VE_i có lực lượng không quá $\frac{n}{k}$ (n-1). Vì mỗi vòng lặp chọn một cặp đỉnh - cạnh thuộc tập VE_i đưa vào tập SE_i, nên số vòng lặp không vượt quá $\frac{n}{k}$ (n-1)

Suy ra thời gian tính toán tính min và cập nhật lại min là: $O(\frac{n^2}{k})$. Thời gian kiểm tra v_{min} ở bước 6 và đưa vào tập S có lực lượng không quá n vậy thời gian tính toán là $O(\frac{n^3}{k})$.

Thời gian liên lạc giữa k bộ xử lý là O(logk). Khi thuật toán kết thúc ta có n bước thông tin liên lạc Vậy tổng thời gian của thuật toán song song là $O(\frac{n^3}{k}) + O(n \log k)$

5. Kết quả thực nghiệm

Cho đồ thị tổng quát như hình 1

Trọng số cạnh w_E được biểu diễn trên đồ thị ở hình 1 và trọng số đỉnh cho ở bảng 1. Áp dụng thuật toán trên tìm đường đi ngắn nhất của cặp đỉnh 1-6 là 1→3→5→6, độ dài 32.

Chúng tôi chạy thuật toán ứng với đồ thị trên trong môi trường phân tán 2 bộ xử lý, trong đó có một bộ xử lý chính (được gọi Server) và 1 bộ xử lý phụ (Client). Bộ xử chính ngoài việc quản lý dữ liệu và phân chia dữ liệu thì còn đảm nhiệm một nhiệm vụ như Client. Chương trình thực nghiệm với bộ xử lý chính (Server) có giao diện như hình 3. Kết quả chương trình thực nghiệm là ổn định và chính xác.

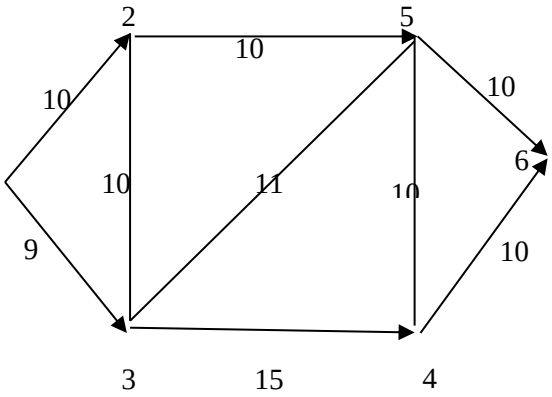
Vertex	Edge 1	Edge 2	w_V
2	(1,2)	(2,3)	1
2	(1,2)	(2,5)	1
2	(3,2)	(2,5)	1
3	(1,3)	(3,4)	1
3	(1,3)	(3,5)	1
3	(1,3)	(3,2)	2
3	(5,3)	(3,2)	1
3	(5,3)	(3,4)	1
3	(2,3)	(3,5)	1
3	(2,3)	(3,4)	1
4	(3,4)	(4,6)	1
4	(3,4)	(4,5)	1
4	(5,4)	(4,6)	1
5	(2,5)	(5,3)	1
5	(2,5)	(5,4)	2
5	(2,5)	(5,6)	3
5	(3,5)	(5,6)	1
5	(3,5)	(5,4)	1
5	(4,5)	(5,3)	1
5	(4,5)	(5,6)	1

Bảng 1. *Trọng số đỉnh*

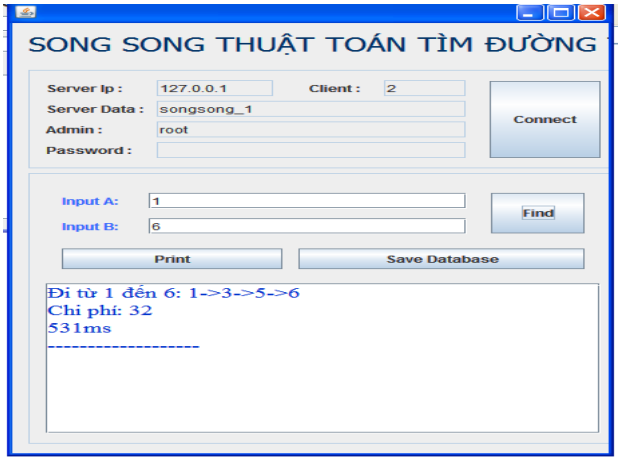
Gọi T_s là thời gian chạy tuần tự, T_p là thời gian chạy song song thì T_s/T_p được gọi là Speedup (mức độ tăng tốc) xem trang 239-252 trong [11]. Mức độ tăng tốc này dùng để đánh giá hiệu năng tính toán của thuật toán song song so với thuật toán tuần tự

Với đồ thị mở rộng tương ứng 100 nút, 150 cạnh và 200 nút-cạnh liên thuộc. Khi xử lý song song trên 2, 4, 6, 8 bộ xử lý khác nhau thì kết quả mô phỏng cho ở hình 5. Kết quả này chứng tỏ thời gian chạy của thuật toán song song tốt hơn so với thuật toán tuần tự tương ứng với đồ thị cố định ở trên. Tuy nhiên điều này không có nghĩa là cứ tăng số bộ xử lý thì thời gian chạy song song sẽ nhanh hơn tuần tự. Với một đồ thị có dữ liệu nhỏ thì đôi lúc thời gian tuyền thông lớn hơn nhiều so với thời gian tính toán trên việc thực hiện song song trên nhiều bộ xử lý khác nhau không làm giảm thời gian tính toán của thuật toán. Hiệu quả của

thuật toán song song được ứng dụng cho các đồ thị đầu vào với dữ liệu lớn mà thời gian xử lý tuần tự thực hiện rất lâu hoặc thậm chí không thực hiện được. Hơn nữa, với đồ thị đầu vào có dữ liệu lớn thì khi tăng số bộ xử lý lên thì thời gian sẽ giảm tương ứng, nhưng đến một lúc nào đó thì thời gian không còn giảm nhanh nữa, thậm chí khi số bộ xử lý quá lớn thì thời gian lại tốn nhiều hơn so với thuật toán chạy trên hệ thống ít bộ xử lý.



Hình 1. *Đồ thị tổng quát G*



Hình 3*. Giao diện Server*

Việc chia 100 nút cho 2 bộ xử lý (tương ứng 4, 6, 8 bộ xử lý) được thực hiện ở bước thứ nhất của thuật toán song song. Việc chia 100 nút cho 2, 4, 6, 8 là đều nhau nghĩa là nếu chia cho 2 bộ xử lý thì mỗi bộ xử lý nhận 50 nút, còn nếu 4 bộ xử lý thì mỗi bộ nhận 25 nút. 6 bộ xử lý thì 5 bộ xử lý đầu nhận 16 nút còn bộ xử lý cuối cùng nhận 20 nút, 8 bộ xử lý thì 7 bộ xử lý đầu nhận 12 nút, bộ xử lý cuối nhận 16 nút. Cách chia tổng quát như sau:

Giả sử có n nút và k bộ xử lý P_0, P_1,…,P_{k-1}

Gọi n_i là số nút của bộ xử lý Pi (i=0,…,k-1).

- Nếu n chia hết cho k thì

For i=0 to k-1 do

n_i=n/k;

- Nếu n không chia hết cho k thì

For i=0 to k-2 do

$$n_i = \left[\frac{n}{k}\right];$$

$$n_{k-1} = n_{k-2} + \left[\frac{n}{k}\right];$$

Tương tự như vậy ta phân trọng số các cung, cách nút - cạnh liên thuộc cho P_i (i=0,…,k-1) có liên quan đến các nút mà bộ xử lý P_i (i=0,…,k-1) đã nhận ở trên.

Ví dụ nếu với đồ thị mở rộng đơn giản 6 nút như trên và chọn 2 bộ xử lý P_0 và P_1 thì chia tập đỉnh T_0={1,2,3} cho P_0, tập đỉnh T_1={4,5,6} cho P_1

Chia tập đỉnh-cạnh liên thuộc VE_0, VE_1 cho P_0, P_1

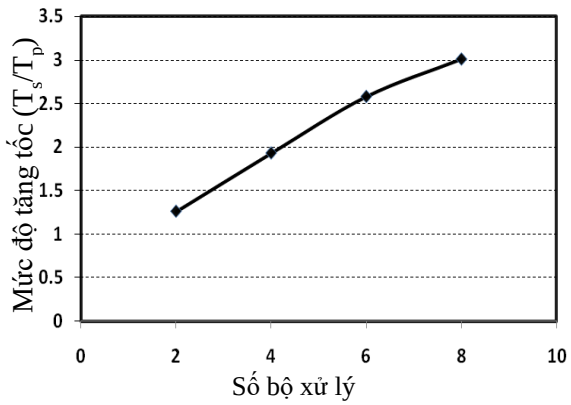
Chia ma trận A_0, A_1 là hai ma trận trọng số cạnh cho hai bộ xử lý P_0 và P_1 (hình 4)

VE_0={ (1, ϕ);(2,(1,2));(2,(3,2)); (3,(1,3); (3,(2,3)); (3,(5,3))}

VE_1={ (4,(3,4)); (4,(5,4)); (5,(2,3)); (5,(3,5)); (5,(4,5)); (6,(5,6)); (6,(4,6))

$$A_0 = \begin{bmatrix} \infty & 10 & 9 \\ \infty & \infty & 10 \\ \infty & 10 & \infty \\ \infty & \infty & \infty \\ \infty & \infty & 11 \\ \infty & \infty & \infty \end{bmatrix} \quad A_1 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 10 & \infty \\ 15 & 11 & \infty \\ \infty & 10 & 10 \\ 10 & \infty & 11 \\ \infty & \infty & \infty \end{bmatrix}$$

Hình 4. *Ma trận trọng số cạnh trên 2 bộ xử lý*



Hình 5. *Mức độ tăng tốc của đồ thị 100 nút, 150 cạnh và 200 đỉnh-cạnh liên thuộc*

6. Kết luận

Trong bài báo này, mô hình đồ thị mở rộng được định nghĩa, thuật toán song song tìm đường đi ngắn nhất giữa hai đỉnh trên đồ thị mở rộng được trình bày cụ thể, có ví dụ thực nghiệm chi tiết và chứng minh được độ phức tạp của thuật toán song song. Chương trình được chạy bằng ngôn Java và quản lý cơ sở dữ liệu MySQL trên hệ thống đa bộ xử lý, dùng mạng LAN.

TÀI LIỆU THAM KHẢO

[1] Trần Quốc Chiến, Nguyễn Mậu Tuệ, Trần Ngọc Việt, "*Thuật toán tìm đường đi ngắn nhất trên đồ thị mở rộng*". Kỷ yếu Hội nghị Khoa học Quốc gia lần thứ VI. FAIR 2013, Huế, 20-21/6/2013.

[2] Nguyễn Đình Lầu, Trần Ngọc việt, *Song song hóa thuật toán dijkstra tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh*, Tạp chí Khoa học, Đại học Huế, Tập 74B, Số 5, 2012, 81-92.

[3] Nguyễn Đình Lầu, Trần Ngọc việt, *Thuật toán song song tìm đường đi ngắn nhất của nhiều cặp đỉnh nguồn đích trên đồ thị*, Tạp chí Khoa học & Công nghệ, Đại học Đà nẵng, quyển 1, số 9 2012, 30-34.

[4] Nguyễn Đình Lầu, Trần Quốc Chiến, Lê Mạnh Thạnh, *Thuật toán song song phân luồng tuyến tính tối ưu trên mạng giao thông mở rộng*, Chuyên san "Các Công trình Nghiên cứu, Phát triển và Ứng dụng Công nghệ Thông tin và Truyền thông Bộ Thông Tin-Truyền Thông, Kỳ 3, Tập V-1, năm 2014.

[5] Trần quốc chiến, Lê Mạnh Thạnh, Nguyễn Đình Lầu, *Thuật toán song song tìm luồng cực đại đồng thời chi phí giới hạn*, kỷ yếu hội thảo quốc gia @16, Những vấn đề chọn lọc của công nghệ thông tin và truyền thông, Đà Nẵng 14-15/11/2013 (accepted).

[6] Nguyễn Đình Lầu, Trần Ngọc Việt, *Song song hóa thuật toán tìm đường đi ngắn nhất của tất cả các đỉnh trên hệ thống cụm máy tính*, Kỷ yếu hội thảo khoa học quốc gia @ lần thứ XV, một số vấn đề chọn lọc của công nghệ thông tin và truyền thông, chủ đề tính toán khoa học, nhà xuất bản khoa học và kỹ thuật, quyết định xuất bản số 152/QĐXB-NXBKHKT, 15/8/1013, trang 403-409

[7] Zilong Ye, An Implementation of Parallelizing Dijkstra's Algorithm, CSE633 Course Project, ID: 3715-8138, 2012

[8] Ned Nedialkov, Parallel Distributed Shortest Paths, SE 3F03, McMaster University Canada, March 2013.

[9] G. Stefano, A. Petricola, C. Zaroliagis, "On the implementation of parallel shortest path algorithms on a supercomputer", in Proc. of ISPA'06, pp. 406-417, 2006

[10] Y. Tang, Y. Zhang, H. Chen, "A Parallel Shortest Path Algorithm Based on Graph-Partitioning and Y. Tang, Y. Zhang, H. Chen, "A Parallel Shortest Path Algorithm Based on Graph-Partitioning and Iterative Correcting", in Proc. of IEEE HPCC'08, pp. 155-161, 2008.

[11] Seyed H. Roosta, *Parallel Processing and parallel Algorithm, Theory ND Computation*, Springer, ISBN 0-387-98716-9, 1999.

(BBT nhận bài: 24/03/2014, phản biện xong: 14/04/2014)