Algoritmos Paralelos para Árvores Geradoras e Componentes Conexos

Edson Norberto Cáceres
Dept. de Matemática - UFMS e COPPE-UFRJ
Caixa Postal 649
79069 - Campo Grande - MS
E-Mail EDSON@BRUFMS.BITNET

RESUMO

Dado um grafo $G = (V, E)$ com n vértices e m arestas, apresentamos uma implementação simples em uma *CREW PRAM* de algoritmos paralelos para computação da árvore geradora e dos componentes conexos de um grafo. Os algoritmos são executados em tempo paralelo $O(\log^2 n)$ com $\frac{m}{\log n}$ processadores. Caso o grafo esteja numerado de forma adequada os algoritmos são ótimos e são executados em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores.

ABSTRACT

Given a graph $G = (V, E)$ with n vertices and m edges, we present a simple implementation in a *CREW PRAM* of parallel algorithms for finding a spanning tree and the connected components of a graph. The algorithms require $O(\log^2 n)$ parallel time with $\frac{m}{\log n}$ processors. If the graph is properly labeled the algorithms are optimal and require $O(\log n)$ parallel time with $\frac{m}{\log n}$ processors.

1 Introdução

A computação de uma árvore geradora e dos componentes conexos de um grafo são problemas básicos em teoria dos grafos e existe uma quantidade considerável de pesquisa no projeto de algoritmos para esses problemas. Os algoritmos seqüenciais usam busca em profundidade ou em largura para resolver eficientemente esses problemas. Os algoritmos paralelos existentes para a computação de uma árvore geradora e dos componentes conexos de um grafo evitam esses procedimentos, visto que não são conhecidos algoritmos paralelos eficientes para busca em profundidade e em largura. Esses algoritmos utilizam a abordagem proposta por *Hirschberg et al.* [10], onde supervértices do grafo são continuamente combinados em supervértices maiores. Esta abordagem foi implementada para uma EREW PRAM [13,12], CREW PRAM [10,4] e CRCW PRAM [1,5,14]. Para um grafo $G = (V, E)$, os algoritmos mais eficientes admitem uma implentação em uma CRCW PRAM em tempo paralelo $O(\log n)$ com $\frac{O((m+n)\alpha(m,n))}{\log n}$ processadores, onde n é o número de vértices, m o número de arestas e $\alpha(m, n)$ é função inversa de Ackermann (cf. [11]).

Um *esteio* [2,3] é uma floresta geradora especial para um grafo bipartido G. Apresentamos algoritmos paralelos que utilizam um esteio para computar a árvore geradora e os componentes conexos de um dado grafo $G = (V, E)$. Estes algoritmos tem como entrada um grafo bipartido. No caso de termos um grafo geral, fazemos uma subdivisão de cada uma das arestas do grafo para que o grafo fique bipartido. As implementações utilizam uma CREW PRAM [11]. Os algoritmos são executados em tempo paralelo $O(\log^2 n)$ com $\frac{m}{\log n}$ processadores, e são de simples implementação visto que são baseados em operações padrões em uma lista, tais como ordenação, compressão e partição, que são descritas em [6,7,8,11].

Se o grafo de entrada estiver rotulado de forma conveniente, os algoritmos são ótimos e podem ser executados em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores. Para a classe dos *st-grafos planares* e grafos *série-paralelo*, essa rotulação pode ser facilmente obtida em tempo $O(\log n)$ com $\frac{m}{\log n}$ processadores.

Na próxima seção apresentaremos as definições utilizadas. A descrição preliminar, detalhes de implementação e exemplos são descritos nas seções seguintes.

2 Notação e Terminologia

O modelo de computação paralela que utilizaremos é a máquina de acesso aleatório paralelo (*PRAM*) [11,7]. Neste modelo de memória compartilhada, em uma unidade de tempo cada processador pode ler uma unidade de informação, escrevê-la, ou executar uma operação aritmética elementar ou uma operação lógica sobre operandos localizados em endereços de memória quaisquer. O modelo PRAM possui diversas variações que diferem na forma de resolver os conflitos de escrita e leitura simultâneas a uma mesma posição de memória por processadores diferentes.

Utilizaremos um modelo PRAM denominado *CREW PRAM* (*Concurrent-Read Exclusive-Write*), esse modelo permite que diversos processadores efetuem a leitura simultaneamente em um mesmo endereço de memória. O comportamento do programa que viola essas restrições é indefinido. Por um longo tempo este parece ter sido o único tipo de PRAM usado.

Os algoritmos que vamos apresentar pertencem à classe *NC* (*Nick Pippenger's Class*) Os problemas pertencentes a essa classe possuem algoritmos determinísticos que são executados em um tempo polilogarítmico usando um número polinomial de processadores. Os problemas da classe *NC* são altamente paralelizáveis, e consideraremos essa classe como sendo a dos algoritmos paralelos eficientes.

Seja $G = (V, E)$ um grafo com n vértices e m arestas.

Um *st-grafo planar* G é um grafo orientado acíclico com exatamente um vértice fonte s e exatamente um vértice sumidouro t, que admite uma representação plana tal que s e t estão na fronteira da face externa.

Seja um grafo G. Dizemos que duas arestas $e_1 = (u, w)$ e $e_2 = (w, v)$ de G estão em *série* se o vértice comum às arestas w tem grau 2. Duas arestas e_1 e e_2 estão em *paralelo* se elas possuem o mesmo conjunto de vértices finais. Definimos uma *composição em série* de uma aresta $e = (u, v)$ como a substituição de e por duas arestas em série, i.e., por (u, w) e (w, v), onde w é um novo vértice. Definimos uma *composição em paralelo* de uma aresta $e = (u, v)$ como a substituição de e por duas arestas e_1 e e_2 que possuam u e v como vértices finais. Uma *redução em série* de duas arestas em série $e_1 = (u, w)$ e $e_2 = (w, v)$ é a substituição de e_1 e e_2 por uma nova aresta $e = (u, v)$. Uma *redução em paralelo* de duas arestas paralelas $e_1 = (u, v)$ e e_2 é a substituição de e_1 e e_2 por uma nova aresta $e = (u, v)$.

Sejam s e t dois vértices de G. Se G pode ser obtido por uma seqüência

de composições em série e em paralelo a partir de uma aresta (s,t), G é denominado um grafo *série paralelo dois terminal* (SPDT) com respeito a s e t. Da mesma forma, G é um grafo SPDT com respeito a s e t se G pode ser reduzido a uma única aresta (s,t) por uma seqüência de reduções em série e em paralelo. Um grafo G é um grafo *série-paralelo* (SP) se existem dois vérices s e t tal que G é um grafo SPDT com respeito a s e t.

As arestas de um grafo SP podem ser orientadas, bastando para isso orientar inicialmente (s,t), e quando nas composições, manter a orientação. O grafo orientado obtido é acíclico [9].

Seja $H = (V_1, V_2, E)$ um grafo bipartido. Se v é um vértice de um subgrafo H' de H, então $d_{H'}(v)$ denota o grau de v em H'. Consideremos os vértices de V_1 ordenados em ordem não crescente com relação a seus graus (em H) e rotulados $u_1, \cdots, u_{|V_1|}$. Seja $\{v_1, \cdots, v_{|V_2|}\}$ os vértices de V_2.

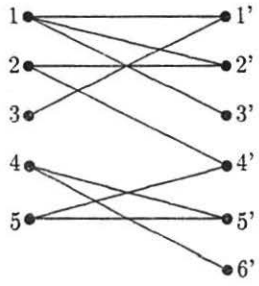
Definimos um *esteio* S *em* V_1 como uma floresta geradora de H tal que cada $v_i \in V_2$ é incidente em S com exatamente uma aresta de E, e (u_j, v_i) é uma aresta de S implica que (u_k, v_i) não é uma aresta de H, para qualquer $u_k \in V_1$, $k < j$.

Para um *esteio em* V_2, as regras para os conjuntos V_1 e V_2, na definição acima são trocadas.

Um vértice $u \in V_1$ é denominado *zero diferença* em S se $d_H(u) - d_S(u) = 0$.

3 Algoritmo para Determinação de uma Árvore Geradora

Nesta seção descrevemos um algoritmo para computação de uma árvore geradora de um grafo bipartido que é implementado em tempo paralelo $O(\log^2 n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM. Caso o grafo esteja rotulado tal que todo vértice $v_i \in V$ tem um adjacente v_j tal que $j > i$ ($j < i$), o algoritmo é ótimo e pode ser executado em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ em uma CREW PRAM. Os grafos st-planares e série-paralelo podem ser facilmente rotulados dessa forma em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM. Usamos o exemplo da Figura 3.1 para ilustrar os passos do algoritmo.



$V_1 = \{1,2,3,4,5\}, V_2 = \{1',2',3',4',5',6'\}$
$E = \{(1,1')(1,2')(1,3')(2,2')(2,4')(3,1')(4,5')(4,6')(5,4')(5,5')\}$

Figura 3.1: Grafo $H = (V_1, V_2, E)$

3.1 Descrição Preliminar do Algoritmo

Algoritmo: Árvore Geradora

1. Computar uma floresta geradora para $H = (V_1, V_2, E)$, obtida com a determinação de um esteio S em H e com a adição de uma aresta arbitrária de $H - S$ a todos vértices não zero diferença de S.

2. Computar um novo grafo bipartido $H_1 = (V_1', V_2', E')$ para representar as conexões existentes entre as diferentes árvores da floresta geradora.

3. Aplicar os passos 1 e 2 até que o número de vértices zero diferença seja igual a um.

3.2 O Algoritmo e um Exemplo

Nesta seção apresentamos o algoritmo para determinação de uma árvore geradora em detalhes juntamente com um exemplo. Os vetores $EDGE$, $EDGE'$, $BEDGE$ e $SPTREE$ contêm m elementos, cada elemento é uma aresta representada por ($vértice1, vértice2$).

Vamos agora descrever o algoritmo. A entrada é o grafo bipartido $H = (V_1, V_2, E)$. A lista de arestas E é armazenada em $EDGE$. Para o nosso exemplo o vetor $EDGE$ é

$$\boxed{(1,1')(1,2')(1,3')(2,2')(2,4')(3,1')(4,5')(4,6')(5,4')(5,5')}$$

Passo 1

Neste passo utilizando um esteio S em V_1, determinamos uma floresta geradora de H. Para a obtenção dos vértices zero diferença e não zero diferença do esteio S, determinamos os graus de cada um dos vértices em V_1 e armazenamos em D_H. Para o nosso exemplo D_H é

$$\boxed{(3,2,1,2,2)}$$

Agora determinamos o esteio S em V_1. Inicialmente efetuamos uma cópia de $EDGE$ em $EDGE'$.

$EDGE' \leftarrow EDGE$

Efetuamos uma ordenação em $EDGE'$ da seguinte formas: Dadas duas arestas (i,j) e (k,l) então $(i,j) < (k,l)$ quando $j < l$ ou $((j = l)$ e $(i < k))$. O vetor $EDGE'$ fica

$$\boxed{(1,1')(3,1')(1,2')(2,2')(1,3')(2,4')(5,4')(4,5')(5,5')(4,6')}$$

Agora comprimimos $EDGE'$ a fim de que nenhum vértice de V_2 apareça mais de uma vez em $EDGE'$. O vetor $EDGE'$ representa S e para o nosso exemplo fica

$$\boxed{(1,1')(1,2')(1,3')(2,4')(4,5')(4,6')}$$

Cada uma das sublistas de $EDGE'$ formadas pelas arestas $(u,v), u = u_i$ forma uma árvore. Rotulamos cada uma dessas árvores por $EDGE'_{u_i}$. Vamos agora determinar quais vértices de V_1 são zero diferença. Para isso, determinamos em paralelo o grau de cada um dos vértices de V_1 em $EDGE'$ e armazenamos em D_S. Para o nosso exemplo

$$\boxed{(3,1,0,2,0,0)}$$

Logo, os vértices zero diferença são

$$\boxed{\{1,4\}}$$

e os não zero diferença são

$$\boxed{\{2,3,5\}}$$

Agora adicionamos uma aresta arbitrária pertencente a $EDGE-S$ a cada um dos vértices não zero diferença ao vetor $EDGE'$. Para o nosso exemplo $EDGE'$ fica

$$\boxed{(1,1')(1,2')(1,3')(2,2')(2,4')(3,1')(4,5')(4,6')(5,4')}$$

Essas arestas adicionadas, conectam as diferentes árvores $EDGE'_{u_i}$ da floresta geradora obtida em S em um nova floresta geradora. Para os vértices não zero diferença $u_j \in V_1$, as árvores $EDGE'_{u_j}$ estão agora conectadas a alguma árvore $EDGE'_{u_k}$ onde u_k é um vértice zero diferença.

O Passo 1 pode ser executado, com a utilização da técnica do posto ótimo de Cole e Vishkin [5] e o algoritmo de ordenacão inteira [8], em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM.

Passo 2

Neste passo construimos um novo grafo bipartido $H_1 = (V_1', V_2', E')$. O grafo H_1 representa as conexões existentes entre as diferentes árvores da floresta geradora representada em $EDGE'$. O grafo H_1 é formado pelas arestas $(u_j, v) \in EDGE-S$ tal que (u_j, v) conecte diferentes árvores $EDGE'_{u_i}$, onde u_i é um vértice zero diferença.

Vamos agora determinar a quais árvores $EDGE'_{u_i}$, u_i um vértice zero diferença, as árvores $EDGE'_{u_j}$, u_j um vértice não zero diferença, estão conectadas. Como demonstraremos posteriormente, a aresta (u_j, v) adicionada após o esteio pode conectar diretamente as árvores, ou fazer parte de um caminho que conecta as árvores.

Para o nosso exemplo, as arestas (u, v) tal que $u = 2, 3$ e 5 estão conectadas à árvore $EDGE'_1$.

Vamos agora determinar a que árvores $EDGE_{u_i}$, u_i um vértice zero diferença, as arestas $e \in EDGE - S$ pertencem. Para o nosso exemplo, as arestas

$$\boxed{(2,2')(3,1')(5,4')}$$

estão conectadas a árvore $EDGE'_1$ e a aresta

$$\boxed{(5,5')}$$

esta conectada a árvore $EDGE'_4$.

Vamos agora construir o grafo bipartido $H_1 = (V'_1, V'_2, E')$. H_1 é formado pelas arestas que conectam diferentes árvores em $EDGE'$. Logo E' é formado pelas arestas (u_j, v_i) e (u_j, v_k), u_j um vértice não zero diferença, tal que essas arestas conectem diferentes árvores. Os conjuntos V'_1 e V'_2 são formados pelos vértice pertencentes as arestas descritas acima. A lista das arestas E' é armazenada no vetor $BEDGE$. Para o nosso exemplo $BEDGE$ é

$$\boxed{(5,4')(5,5')}$$

O passo 2 pode ser executado, com a utilização da técnica do posto ótimo de Cole e Vishkin [5], em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM.

Passo 3

Neste passo aplicamos os passos 1 e 2 até que o número de vértices zero diferença (*numzerodif*) seja igual a 1. Isso pode ser efetuado da seguinte maneira.

repita
  $SPTREE \leftarrow SPTREE + EDGE'$
  $EDGE \leftarrow BEDGE$
ate $|numzerodif| = 1$

Após isso, retiramos as arestas duplicadas em $SPTREE$.

Como veremos, o número de vértices zero diferença ao final do Passo 2, é pelo menos dividido por dois a cada iteração. Portanto no Passo 3, o algoritmo executa no máximo $\lceil \log n \rceil$ iterações.

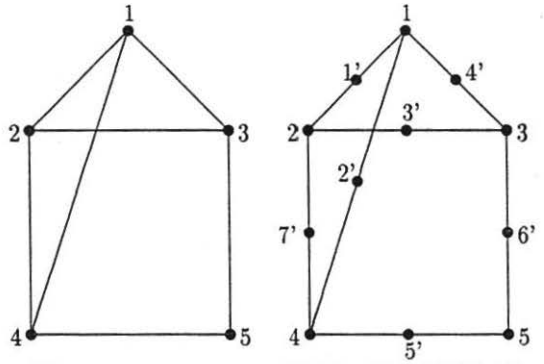


Figura 4.2: Grafos $G = (V, E)$ e $H = (V, V', E')$

4 Grafos Não Orientados

Para grafos não orientados usamos o mesmo algoritmo, efetuando o seguinte passo adicional. Realizamos a subdivisão de cada aresta, e com isso colocamos um vértice adicional em cada aresta. Com isso, o grafo não orientado resultante é um grafo bipartido. Podemos considerar apenas uma das arestas (u, v) ou (v, u), $v \in V'$, visto que com a subdivisão, todos os vértices adicionados serão considerados no conjunto de vértices V'.

Dessa forma, dado um grafo não orientado $G = (V, E)$, substituimos as arestas $(i, j) \in E$ por duas arestas (i, k) e (k, j), $k \in V'$. O grafo obtido $H = (V, V', E')$ é um grafo bipartido. Logo podemos aplicar o algoritmo da seção 2 para obter uma árvore geradora para H.

Ilustramos os passos principais do algoritmo com relação ao grafo da Figura 4.2.

$$\boxed{(1,2)(1,3)(1,4)(2,1)(2,3)(2,4)(3,1)(3,2)(3,5)(4,1)(4,2)(4,5)(5,3)(5,4)}$$

$$\boxed{(1,1')(1,2')(1,4')(2,1')(2,3')(2,7')(3,3')}$$

$$\boxed{(3,4')(3,6')(4,2')(4,5')(4,7')(5,5')(5,6')}$$

Aplicamos o algoritmo em H e obtemos uma árvore geradora. Ao final do algoritmo $SPTREE$ armazena a árvore geradora de H. Vamos agora computar a árvore geradora de $G = (V, E)$. Isso pode ser feito através da computação do grau de cada um dos vértices $v_i \in V'$. Seja $SPTREE'$ as

arestas de $SPTREE$ tal que $grau(v_i) = 2$. Temos que $SPTREE'$ é uma árvore geradora de G. Para o nosso exemplo, temos que $SPTREE$ é

$$\boxed{(1,1')(1,2')(1,4')(2,1')(2,3')(2,7')(3,3')(3,6')(4,2')(4,5')(5,5')}$$

e $SPTREE'$ é

$$\boxed{(1,2)(1,4)(2,3)(4,5)}$$

5 Componentes Conexos

Nesta seção descreveremos brevemente como o algoritmo para a computação de uma árvore geradora pode ser aplicado para determinar os componentes conexos de um grafo e os componentes fracamente conexos de um grafo orientado. Estes problemas podem ser resolvidos em tempo paralelo $O(\log^2 n)$ com $\frac{m}{\log n}$ processadores e em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores quando os vértices do grafo estiverem convenientemente rotulados.

O algoritmo para determinação de uma árvore geradora conecta a cada iteração as árvores $EDGE'_{u_i}$. Ao final do algoritmo, representamos cada árvore com o menor vértice. Cada um dos diferentes vértices representa um componente conexo do grafo.

Os componentes fracamente conexos do grafo orientado podem ser obtidos com a desconsideração das orientações das arestas e com a remoção das arestas duplicadas.

6 Uma Simplificação do Algoritmo

O algoritmo para computação de uma árvore geradora para um grafo $G = (V, E)$ pode ser bastante simplificado. Essa simplificação é no sentido de evitar as ordenações que são efetudas nos passos do algoritmo.

Ao computarmos o esteio no grafo bipartido $H = (V_1, V_2, E)$, não necessitamos ordenar os vértices. Podemos selecionar para cada uma das arestas saindo dos vértices $v \in V_2$, a aresta que chegue ao vértice de maior grau de V_1. Ao escolhermos os vértices de maior grau para selecionar as arestas, estamos apenas diminuindo o número de iterações necessárias para a computação de uma árvore geradora.

7 Certificação e Análise

Lema 7.1 *Seja A o conjunto das arestas que foram adicionadas aos vértices não zero diferença u_j, $EDGE' = S \bigcup A$ e $EDGE'_{u_i}$ a sublista das arestas $(u, v) \in EDGE'$ tal que $u = u_i$. Então a aresta $e \in A \bigcap EDGE'_{u_j}$ conecta a árvore $EDGE'_{u_j}$ com a árvore $EDGE_{u_i}$, u_i um vértice zero diferença ou existe um caminho $(u_j, v_k), \cdots, (v_l, u_i)$ ligando $EDGE'_{u_j}$ a $EDGE'_{u_i}$.*

Demonstração: Utilização de duplicação recursiva. □

Lema 7.2 *Seja $H = (V_1, V_2, E)$ um grafo bipartido conexo e S um esteio em V_1. Seja $H' = (V_1, V_2, E_1)$ o grafo obtido de H adicionando precisamente a S uma aresta de $E - S$ incidente a cada vértice não zero diferença de V_1. Então H' é acíclico. Além disso, se V_1 contém exatamente k vértices zero diferença então H' é uma floresta geradora de H com k árvores.*

Demonstração: S é essencialmente uma coleção de estrelas cujos centros são vértices em V_1. Adicionando precisamente uma aresta de $E - S$ incidente a cada vértice não zero diferença de V_1, cada estrela cujo centro é um vértice não zero diferença v_j será conectada com no máximo uma estrela distinta cujo centro é v_i. Então H' é acíclico.

Utilizando o lema acima, todas estrelas cujo centro é um vértice não zero diferença estarão conectadas por uma aresta ou por um caminho a uma estrela cujo centro é um vértice zero diferença. Logo para cada um dos k vértices zero diferença teremos uma árvore.□

Teorema 7.1 *Seja $H_1 = (V'_1, V'_2, E')$ um grafo bipartido representando as conexões existentes entre as diferentes árvores da floresta geradora obtida ao final do passo 2. O número de vértices zero diferença de H_1 é no máximo $\lceil \frac{|V'_2|}{2} \rceil$.*

Demonstração: Temos que o esteio seleciona uma única aresta saindo de $v_i \in V'_2$, esta aresta conecta v_i a um vértice $u_j \in V'_1$, tal que o grau de u_j é o maior entre os vértices que estão conectados a v_i. Logo o grau de um vértice $u_j \in V'_1$ zero diferença é pelo menos dois, e o número de vértices zero diferença é no máximo $\lceil \frac{|V'_2|}{2} \rceil$.□

Portanto, no Passo 3, o algoritmo itera no máximo $O(\log n)$ vezes no pior caso.

Teorema 7.2 *O algoritmo determina uma árvore geradora para um grafo em tempo paralelo* $O(\log^2 n)$ *com* $\frac{m}{\log n}$ *processadores em uma CREW PRAM.*

Vamos agora demonstrar os resultados, que são obtidos caso o grafo esteja convenientemente rotulado.

Lema 7.3 *Seja* $G = (V, E)$ *um grafo tal que os vértices de* G *estão rotulados de acordo com uma das duas condições:* (i) *para todo* $1 < v_i \leq n$, v_i *tem um adjacente* v_j *tal que* $j < i$; (ii) *para todo* $1 \leq v_i < n$, v_i *tem um adjacente* v_j *tal que* $j > i$. *Então o grafo bipartido* H *possui apenas um vértice zero diferença.*

Demonstração: Vamos considerar o caso (i), o (ii) pode ser demonstrado de forma análoga. Na computação de um esteio para o grafo bipartido obtido de G, para cada um dos vértices obtidos com a subdivisão, selecionamos as arestas incidentes com os vértices de maior rótulo. Logo as arestas que saem do vértice de maior rótulo v_k são selecionadas. Em função de que para todo vértice $v_i \neq v_k$ tem pelo menos um vértice adjacente v_j, $j > i$, pelo menos uma aresta saindo de v_i que não será selecionada. Logo, apenas o vértice v_k será um vértice zero diferença.□

Portando, nesse caso em uma única iteração o algoritmo computa uma árvore geradora.

Teorema 7.3 *Se os vértices do grafo* G *estiverem rotulados de acordo com o lema anterior, o algoritmo computa uma árvore geradora em tempo paralelo* $O(\log n)$ *com* $\frac{m}{\log n}$ *processadores em uma CREW PRAM.*

Teorema 7.4 *Seja* $G = (V, E)$ *um grafo e uma orientação acíclica de* G. *Se o grafo orientado resultante da orientação contiver exatamente uma fonte ou exatamente um sumidouro então o algoritmo computa uma árvore geradora de* G *em tempo paralelo* $O(\log n)$ *com* $\frac{m}{\log n}$ *processadores em uma CREW PRAM.*

Demonstração: Vamos considerar o caso de obtermos uma orientação com apenas uma fonte. Podemos efetuar uma ordenação dos vértices de acordo com a orientação e rotular os vértices de acordo com a ordenação resultante. Logo para todo vértice $1 < v_i \leq n$, v_i tem um adjacente v_j tal que $j < i$. Isso faz com que o algoritmo computa uma árvore geradora de G em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$.□

Teorema 7.5 *Seja $G = (V, E)$ um grafo st-planar, então o algoritmo computa uma árvore geradora para G em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM.*

Teorema 7.6 *Seja $G = (V, E)$ um grafo série paralelo, então o algoritmo computa uma árvore geradora para G em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores em uma CREW PRAM.*

8 Conclusões

Utilizando esteio, apresentamos algoritmos paralelos de simples implementação para a computação de uma árvore geradora e dos componentes conexos de um grafo. As implementações utilizam uma CREW PRAM. Os algoritmos são executados em tempo paralelo $O(\log^2 n)$ com $\frac{m}{\log n}$ processadores. Se o grafo de entrada estiver rotulado convenientemente os algoritmos são ótimos e podem ser executados em tempo paralelo $O(\log n)$ com $\frac{m}{\log n}$ processadores. Para a classe dos *st-grafos planares* e grafos *série-paralelo*, essa rotulação pode ser facilmente obtida em tempo $O(\log n)$ com $\frac{m}{\log n}$ processadores.

Acreditamos que outras classes de grafos possuem algoritmos paralelos ótimos. Para isso, os vértices do grafo necessitam ser convenientemente rotulados, ou que possuam uma orientação com uma única fonte ou com um único sumidouro, e que a rotulação ou a orientação possam ser efetuadas em tempo paralelo menor que $O(\log^2 n)$ em uma CREW PRAM.

Referências

[1] B. Awerbuch and Y. Shiloach. New connectivity and MSF algorithms for shuffle-exchange network and PRAM. *IEEE Transactions on Computer*, C-36:1258–1263, 1987.

[2] E.N. Cáceres, N. Deo, S. Sastry, and J.L. Szwarcfiter. Parallel algorithm for Euler tour. Technical report, University of Central Florida, 1990.

[3] E.N. Cáceres, N. Deo, S. Sastry, and J.L. Szwarcfiter. On finding Euler tours. *Parallel Proc. Letters*, a ser publicado, 1992.

[4] F.Y. Chin, J. Lam, and I. Chen. Efficient parallel algorithms for some graph problems. *Communication of ACM*, 25:659–665, 1982.

[5] R. Cole and U. Vishkin. Approximate and exact parallel scheduling with applications to optimal parallel list ranking. *IEEE Annual Symposium on Foundations of Computer Science*, pgs. 478–491, 1986.

[6] D. Eppstein and Z. Galil. Parallel algorithmic techniques for combinatorial computation. *Annals Review Computer Science*, 3:233–283, 1988.

[7] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[8] T. Hagerup and H. Shen. Improved nonconservative sequential and parallel integer sorting. *Information Processing Letters*, 36:57–63, 1990.

[9] X. He. Efficient parallel algorithms for series parallel graphs. *J. Algorithms*, 12:409–430, 1991.

[10] D.S. Hirschberg, A.K. Chandra, and D.V. Sarwate. Computing connected components on parallel computers. *Communication of ACM*, 22:461–464, 1979.

[11] R.M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science- Vol. A*, Capítulo 17, pgs. 869–941. The MIT Press/Elsevier, 1990.

[12] C.P. Kruskal, L. Rudolph, and M. Snir. Efficient parallel algorithms for graph problems. *Algorithmica*, 5:43–64, 1990.

[13] D. Nath and N. Maheshwari. Parallel algorithms for the connected components and minimal spanning tree problems. *Information Processing Letters*, 14:7–11, 1982.

[14] Y. Shiloach and U. Vishkin. An $o(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–63, 1982.