Parallel Recognition and Location Algorithms for Chordal Graphs using Distance Matrices

Stavros D. Nikolopoulos
Department of Computer Science, University of Cyprus,
75 Kallipoleos Str., P.O.Box 537, Nicosia, Cyprus.
stavros@jupiter.cca.ucy.cy

Abstract. We present efficient parallel algorithms for recognizing chordal graphs and locating all maximal cliques of a chordal graph G=(V,E). Our techniques are based on partitioning the vertex set V using information contained in the distance matrix of the graph. We use these properties to formulate parallel algorithms which, given a graph G=(V,E) and its adjacency-level sets, decide whether or not G is a chordal graph, and, if so, locate all maximal cliques of the graph in time $O(k)$ by using $\delta^2 \cdot n^2/k$ processors on a CRCW-PRAM, where δ is the maximum degree of a vertex in G and $1 \le k \le n$. The construction of the adjacency-level sets can be done by computing first the distance matrix of the graph, in time $O(\log n)$ with $O(n^{\beta}+D_G)$ processors, where D_G is the output size of the partitions and $\beta=2.376$, and then extracting all necessary set information. Hence, the overall time and processor complexity of both algorithms are $O(\log n)$ and $O(\max\{\delta^2 \cdot n^2/\log n, n^{\beta}+D_G\})$, respectively. These results imply that, for $\delta \le \sqrt{n \log n}$, the proposed algorithms improve in performance upon the best-known algorithms for these problems.

Keywords. Parallel algorithms, Chordal graphs, Recognition, Maximal cliques, Distance matrix, Graph partition, Complexity.

1 Introduction

A graph G=(V,E) is called *chordal* (or triangulated) if every cycle of length, at least, four has a chord, i.e., an edge joining two nonconsecutive vertices of the cycle. Triangulated graphs arise in the study of Gaussian elimination on sparse symmetric matrices [14, 15], in the study of acyclic relational schemes [2], and are related to and useful for many location problems [8, 9].

Our objective is to study the parallel recognition of a chordal graph, as well as the parallel location of all maximal cliques of such a graph. Fulkerson and Gross [7] suggested an iterative procedure to recognize chordal graphs and pointed out a property of the maximal cliques of a chordal graph [8]. Edenbrandt [5] proposed a parallel algorithm for recognizing chordal graphs which can be executed in time $O(\log n)$ with $O(n^5)$ processors on a CRCW-PRAM or in time $O(\log^2 n)$ with $O(n^5)$ processors on a CREW-PRAM. Naor, Naor and Schäffer [13] proposed a parallel recognition algorithm which runs in time $O(\log^2 n)$ by using $O(n^3)$ processors on a CREW-PRAM. They also proposed parallel algorithms for some other problems on chordal graphs among which a parallel algorithm for computing all maximal cliques which runs in time $O(\log^3 n)$ using $O(n^4)$ processors or in time $O(\log^2 n)$

using $O(n^5)$ processors on the same type of computational model. After the publication of Naor, et. al. [13], Dahlhaus and Karpinski [4] proposed a new parallel algorithm for finding maximal cliques in time $O(log^2 n)$ using $O(n^4)$ processors on a CREW-PRAM. Klein [11] has announced efficient parallel algorithms for several problems on chordal graphs, which run in time $O(log^2 n)$ using $O(n+m)$ processors on a CRCW-PRAM, where m is the number of edges in the graph. Ho and Lee [9] formulated an algorithm which computes a clique tree in time $O(log n)$ with $O(n^3)$ processors on a CRCW-PRAM. Subsequently these authors [8] formulated an algorithm which, given a clique tree of a graph, computes a perfect vertex elimination scheme in time $O(log n)$ with $O(n^2)$ processors in the same type of computational model. This implies that a chordal graph can be recognized in time $O(log n)$ with $O(n^3)$ processors on a CRCW-PRAM. Moreover, these authors [9] proposed an algorithm which computes all maximal cliques of a chordal graph in time $O(log n)$ on a CRCW-PRAM or in time $O(log^2 n)$ on a CREW-PRAM using $O(n^3)$ processors.

In this paper we present efficient parallel algorithms for the problems of recognising a chordal graph and finding all maximal cliques of a chordal graph. We start with the notion of partitioning the vertex set V of a graph G=(V,E), with respect to a vertex $v \in V$, into a set of (mutually disjoint) adjacency-level sets AL(v, 0), AL(v, 1), ..., AL(v, L), $0 \le L < n$, using the information contained in the distance matrix of the graph. We show the following properties of the adjacency-level sets of a chordal graphs G=(V,E):

(i) The vertex set $adj(x) \cap AL(v,\ell-1)$ is a clique, for every $x \in AL(v,\ell)$, $1 \le \ell \le L$.

(ii) If $(x,y) \in E$, then either $adj(x) \cap AL(v,\ell-1) \subseteq adj(y) \cap AL(v,\ell-1)$ or $adj(y) \cap AL(v,\ell-1) \subseteq adj(x) \cap AL(v,\ell-1)$, for every $x,y \in AL(v,\ell)$, $1 \le \ell \le L$.

(iii) If a vertex set C is a maximal clique of the graph G, then the graph G(C) induced by C is a subgraph of the graph $G(AL(v,\ell-1) \cup AL(v,\ell))$,

(iv) Given a vertex $x \in AL(v,\ell)$, a maximal clique of the graph G containing the vertex x has the form

$$\{adj(x) \cap AL(v,\ell-1)\} \cup C_{x,y}$$

where $y \in adj(x) \cap AL(v,\ell)$, $1 \le \ell \le L$, and $C_{x,y}$ is a clique in graph $G(AL(v,\ell))$ containing vertices x and y.

Subsequently, based on these properties, we formulate parallel algorithms which, given a graph G=(V,E) and its adjacency-level sets, solve the problems mentioned above, directly, in time $O(k)$ by using $\delta^2 \cdot n^2/k$ processors on a CRCW-PRAM, where δ is the maximum degree of a vertex in G and $1 \le k \le n$.

For the process of partitioning, we first compute the distance matrix D of a graph by using the parallel algorithm in [3] which runs in time $O(log n)$ with $O(n^{\beta}+D_G)$ processors, where $\beta=2.376$ and D_G is the output size of the partitions of the graph. Then, given the distance matrix of a graph, it is possible to construct the adjacency-level sets $AL(v,\ell)$, $0 \le \ell \le L_v$, in constant time $O(1)$ using $O(n)$ processors on a CRCW-PRAM.

It turns out that, the overall time and processor complexity of both algorithms proposed in this paper are $O(log n)$ and $O(\max\{\delta^2 \cdot n^2/log n, n^{\beta+D_G}\})$, respectively. These results imply that the proposed algorithms run in time $O(log n)$ and have a total cost of $O(\max\{\delta^2 \cdot n^2, log n \cdot (n^{\beta+D_G})\})$. The best-known parallel algorithms for the recognition and all maximal clique location problems run in time $O(log n)$ by using $O(n^3)$ processors [9, 10] or in time $O(log^2 n)$ by using (n+m) processors [11]. Therefore, for $\delta \leq \sqrt{n log n}$, we improve upon the best-known algorithms for these problems in performance. Moreover, some other advantages of our algorithms over previous ones are:
(i) their correctness proof is simple, (ii) they avoid computing first a number of other entities such as clique trees or perfect elimination schemes, and (iii) they work with the same number of processors in constant time $O(1)$ on a CRCW-PRAM computational model if the distance matrix of the graph is given.

2 Adjacency-Level Sets and Distance Matrices

Given a graph G=(V, E) and a vertex $v \in V$, we define a partition $\mathcal{L}(G,v)$ of the vertex set V (we shall frequently use the term *partition of the graph* G), with respect to the vertex v, as follows:

$$\mathcal{L}(G,v) = \{ AL(v,\ell) \mid v \in V, 0 \leq \ell \leq L_v, 1 \leq L_v < |V| \}$$

where $AL(v,\ell)$, $0 \leq \ell \leq L_v$, are the *adjacency-level sets*, and L_v is the *length* of the partition $\mathcal{L}(G,v)$. The adjacency-level sets of the partition $\mathcal{L}(G,v)$, are defined as follows:

$$AL(v,\ell) = \{ w \mid d(v,w) = \ell,\ 0 \leq \ell \leq L_v \}$$

where d denotes the *minimum distance* in G. Note that $d(v,w) \geq 0$ and $d(v,w)=0$ iff v=w. By definition, the adjacency-level sets have the following properties:

$$AL(v,\ell) \cap AL(v,\ell') = \emptyset \quad \text{for } \ell \neq \ell'$$

and

$$\bigcup_{0 \leq \ell \leq L_v} AL(v,\ell) = V$$

Moreover,

$$adj(x) \cap AL(v,\ell-1) \neq \emptyset \qquad \forall\, x \in AL(v,\ell),\ 1 \leq \ell \leq L_v$$
$$adj(x) \cap AL(v,\ell-2) = \emptyset \qquad \forall\, x \in AL(v,\ell),\ 2 \leq \ell \leq L_v.$$

The computation of the adjacency-level sets can easily be done by considering first the distance matrix of the graph G and then extracting all set information that is necessary. Let D be the distance matrix of G. Then, the vertex set $AL(v,\ell)$ containts the vertex x if and only if $D[v,x] = \ell$, $1 \leq \ell \leq L_v$. Then, the vertex set $AL(v,\ell)$, $1 \leq \ell \leq L_v$, can be constructed as follows: Set $AL(v,\ell) \leftarrow x$, if $D[v,x] = \ell$, for every $x \in V$.

3 Properties

The following lemmas provide some properties of the adjacency-level sets of a chordal graph leading to a fast parallel recognition algorithm and to a fast parallel maximal clique location algorithm.

Lemma 1. *Consider a chordal graph G=(V,E) and its adjacency-level sets AL(v,0), AL(v,1), ..., AL(v,L_v), v∈V. Let x be a vertex of set AL(v,ℓ), 1≤ℓ≤L_v. The vertex set adj(x)∩AL(v,ℓ-1) is a clique.*

Proof. We assume that adj(x)∩ AL(v,ℓ -1) is not a clique. Then there exist vertices u_1, u_2∈adj(x)∩AL(v,ℓ -1) such that (u_1,u_2)∉E, (u_1,x)∈E and (u_2,x)∈E. Therefore, graph G(AL(v,0) ∪ ...∪ AL(v,ℓ -1)∪ AL(v,ℓ)) contains a cycle [v, ..., u_1, x, u_2, ..., v] of length greater than 3, which is absurd. □

Lemma 2. *Consider a chordal graph G=(V,E) and its adjacency level sets AL(v,0), AL(v,1), ..., AL(v,L_v), v∈V. Let x, y be vertices of AL(v,ℓ), 1≤ℓ≤L_v. If (x,y)∈E, then one of the following alternatives holds*:

(i) $adj(x)\cap AL(v,\ell-1) \subseteq adj(y)\cap AL(v,\ell-1)$

(ii) $adj(y)\cap AL(v,\ell-1) \subseteq adj(x)\cap AL(v,\ell-1)$.

Proof. (i) Let |adj(x)∩AL(v,ℓ -1)| ≥ |adj(y)∩AL(v,ℓ -1)|, where x,y∈AL(v,ℓ). Assume that adj(x)∩AL(v,ℓ -1) ⊂ adj(y)∩AL(v,ℓ -1). Then there exists vertex u_1∈adj(y)∩AL(v,ℓ -1) such that u_1∉adj(x)∩AL(v,ℓ -1), i.e., (u_1,y)∈E and (u_1,x)∉E. Since |adj(x)∩AL(v,ℓ -1)|≥ |adj(y)∩AL(v,ℓ -1)|, there follows that there is a vertex u_2∈adj(x)∩AL(v,ℓ -1) such that (u_2,x)∈E, (u_2,y)∉E and (u_1,u_2)∉E. Therefore, we are led to the conclusion that there is a cycle [u_1, y, x, u_2, ..., u_1] in graph G(AL(v,0) ∪ ...∪AL(v,ℓ -1)∪AL(v,ℓ)) of length greater than 3, which is absurd. Thus, adj(x)∩AL(v,ℓ -1) ⊆ adj(y)∩AL(v,ℓ -1).
(ii) Let |adj(x)∩AL(v,ℓ -1)| ≤ |adj(y)∩AL(v,ℓ -1)|. In an similar way it is shown that adj(y)∩AL(v,ℓ -1) ⊆ adj(x)∩AL(v,ℓ -1). □

4 Parallel Recognition

The recognition algorithm proposed here is simple and based on the results provided by Lemma 1 and Lemma 2, as well as on the following observation: Let G=(V,E) be an undirected graph which contains a cycle C of length grater than 3. Let v∈V be a vertex of the cycle C, i.e., v∈C. If we partition the graph G, with respect to vertex v∈C, then exactly one of the following alternatives holds:

(i) There exists a vertex x∈AL(v,ℓ), 1<ℓ ≤ L_v, such that the set adj(x)∩AL(v,ℓ -1) is not a clique (see lemma 1).

(ii) There exists vertices x,y∈ AL(x,ℓ), 1<ℓ ≤ L_v, such that adj(y)∩AL(x, ℓ -1) ≠ adj(z)∩AL(x, ℓ -1), (see lemma 2).

Obviously, (i) holds if the length of the cycle C is even, while (ii) holds if the length of the cycle C is odd.

Based on the previous facts and observation, we formulate a parallel algorithm for determining whether or not a graph is a chordal graph. The correctness of the algorithm follows directly from Lemmas of Section 3. The recognition algorithm is listed in fig. 1.

Algorithm RECOGNITION
input : Graph G=(V, E) and the a.l.s AL(v, 0), AL(v, ℓ), ..., AL(v, L_v), v∈V, of G.
output : Is G chordal graph?
begin
1. **for** each v∈V **do in parallel**
1.1. **for** each x ∈ AL(v, ℓ), $1 \le \ell \le L_v$, **do in parallel**
$S_{v,x} \leftarrow$ adj(x) ∩ AL(v, ℓ -1);
end;
1.2. **for** each x ∈ AL(v, ℓ), $1 \le \ell \le L_v$, **do in parallel**
1.2.1. **if** $S_{v,x}$ is not a clique **then return** "G is not a chordal graph";
1.2.2. **for** each y ∈ adj(x) ∩ AL(v, ℓ), $1 \le \ell \le L_v$, **do in parallel**
if $S_{v,x} \neq S_{v,y}$ **then return** "G is not a chordal graph";
end;
end;
end;
2. **return** "G is a chordal graph";
end.

Fig. 1. The Parallel Recognition Algorithm RECOGNITION

4.1 The Complexity of the Recognition Algorithm

Having proved the correctness of algorithm RECOGNITION, let us now analyse its complexity. We shall obtain the overall computational complexity of the algorithm by computing the complexity of each step separately. We take our processors to be Concurrent-Read Concurrent-Write (CRCW) PRAM [1, 6, 12, 16].

Step 1. This step consists of two substeps, i.e., substeps 1.1 and 1.2, which are executed sequentially. In substeps 1.1 the intersection of n-1 pairs of vertex sets are computed. The number of vertices in set adj(x), x∈V, is δ, where δ is the maximum degree of a vertex in G=(V,E). Therefore, this substeps is executed in constant time $O(1)$ using $\delta \cdot (n-1)$ processors. Substeps 2.2.1 checks n vertex sets for completeness. This can be done in constant time $O(1)$ with $\delta^2 \cdot n$ processors. Substeps 2.2.2 checks $\delta \cdot n$ pairs of vertex sets to determine whether or not there is a pair with not equal elements. This operation can be executed in constant time $O(1)$ with $\delta^2 \cdot n$ processors. Clearly, step 2 is executed in constant time $O(1)$ with $\delta^2 \cdot n^2$ processors. Moreover, from the above analysis, it is easy to see that this step can also be executed in time $O(\log n)$ with $\delta^2 \cdot n^2 / \log n$ processors.

The computation of the adjacency-level sets can easily be done by considering first the distance matrix of the graph G and then extracting all set information that is necessary. This

can be done in time $O(\log n)$ with $O(n^{\beta}+D_G)$ processors with matrix multiplication techniques [2] for β=2.376, where D_G is the output size of the partitions $\mathcal{L}(G,v)$, for all $v \in V$.

Thus, given the adjacency-level sets of a graph G=(V,E), with respect to every vertex $v \in V$, the algorithm RECOGNITION runs in time $O(\log n)$ when $\delta^2 \cdot n^2/\log n$ processors are available on a CRCW-PRAM model. Consequently, we can formulate the following theorem.

Theorem 3. *A chordal graph G=(V,E) can be recognized in time $O(\log n)$ with $O(\max\{\delta^2 \cdot n^2/\log n, n^{\beta}+D_G\})$ processors on a CRCW-PRAM model.*

5 Parallel Location of all Maximal Cliques

Let G=(V, E) be a chordal graph and let AL(v, 0), AL(v, 1), ..., AL(v, L_v) be the adjacency-level sets of a partition $\mathcal{L}(G,v)$, $v \in V$. We consider an arbitrary vertex $x \in AL(v,\ell)$, $1 \le \ell \le L_v$. The following statements hold:

(i) Graph $G(\{x\} \cup \{adj(x) \cap AL(v,\ell-1)\})$ is a clique, (see lemma 1).

(ii) If $x, y \in AL(v,\ell)$ and $(x,y) \in E$, then
$adj(x) \cap AL(v,\ell-1) \subseteq adj(y) \cap AL(v,\ell-1)$ or
$adj(y) \cap AL(v,\ell-1) \subseteq adj(x) \cap AL(v,\ell-1)$, (see lemma 2).

(iii) If C is a maximal clique in graph G=(V,E), then $C \subseteq AL(v,\ell) \cup AL(v,\ell-1)$, $1 \le \ell \le L_v$, (see properties of the adjacency-level sets of G).

Next we define four sets of vertices for each vertex $x \in AL(v,\ell)$, $1 \le \ell \le L_v$, which we denote by B_x, F_x, W_x and $C_{x,y}$, where $y \in AL(v,\ell)$.

B_x is defined to be the set that contains all the vertices of $AL(v,\ell-1)$ which join vertex $x \in AL(v,\ell)$ by an edge, i.e.,

$$B_x = \{y \mid y \in adj(x) \cap AL(v,\ell-1)\}.$$

F_x is defined to be the set that contains all the vertices of $AL(v,\ell)$ which join every vertex of B_x by an edge, i.e.,

$$F_x = \{y \mid y \in AL(v,\ell) \ \& \ (y,z) \in E, \ \forall z \in B_x\}.$$

W_x is defined to be the set that contains all the vertices of F_x which join vertex $x \in F_x$ by an edge, i.e.,

$$W_x = \{y \mid y \in F_x \ \& \ (y,x) \in E\}.$$

$C_{x,y}$ is defined as the set which contains the vertex $x \in AL(v,\ell)$, the vertex $y \in W_x$ and all the vertices of W_x which join vertices x and y by an edge, i.e.,

$$C_{x,y} = \{x,y\} \cup \{adj(x) \cap adj(y) \cap W_x\}.$$

Notice that $C_{x,x}=\{x\}$. Clearly, if $C_{x,y}$ is a clique in $G(AL(v,\ell))$ which contains vertices x and y, then $C_{x,y}$ is a maximal clique in $G(AL(v,\ell))$. It is also clear that graph

G(AL(v,ℓ)) may contain more than one maximal cliques which contain vertices x and y.

In fig. 2(a) $B_x=\{u_1, u_2\}$, $F_x=\{y\}$, $W_x=\{y\}$, $C_{x,y}=\{x, y\}$ and $B_y=\{u_1, u_2, u_3, u_4\}$, $F_y=\emptyset$, $W_y=\emptyset$, $C_{y,y}=\{y\}$. In fig. 2(a) there is one maximal clique which contains vertices x and u, i.e., C={x,u}, while in fig. 2(b) there are two such maximal cliques, i.e., C={x, u, y} and C′ ={x, u, z}.

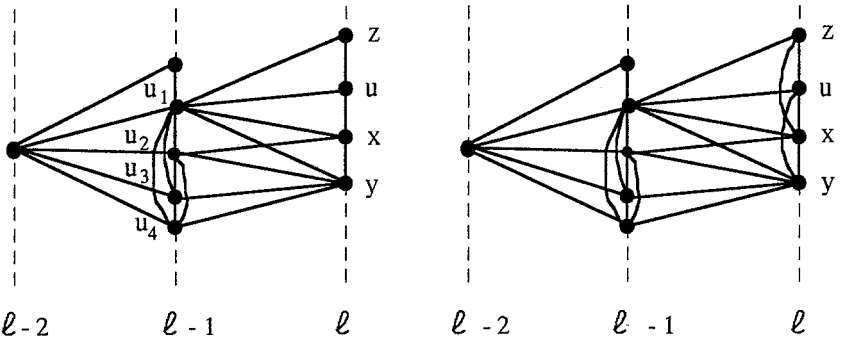

Fig. 2(a) & 2(b). Three consecutive adjacency-level sets of a chordal graph

The following Lemma shows that we can always compute a maximal clique $C_{x,y}$ of the graph G(AL(v,ℓ)), $x\in AL(v,\ell)$ and $y\in W_x$, $1\le\ell\le L_v$.

Lemma 4. *Let G=(V,E) be a chordal graph, and let AL(v,0), AL(v,1), ..., AL(v,L_v) be the adjacency-level sets of $\mathcal{L}$(G,v), v∈V. Let C be a maximal clique of graph G(AL(v,ℓ)), $1\le\ell\le L_v$, which contains the vertex x. Then, there exists a vertex y∈C such that* $\{x,y\}\cup\{adj(x)\cap adj(y)\cap AL(v,\ell)\}=C$, $1\le\ell\le L_v$.

Proof. If |C|=1 then there is nothing to prove. Consider the case where |C|>1. Since vertex x∈C and vertex set C is a maximal clique in graph G(AL(v,ℓ)), $1\le\ell\le L_v$, there follows that $C\subseteq\{x\}\cup\{adj(x)\cap AL(v,\ell)\}$ (see fig. 2(b), C={x, y, u}). Let $z\in\{x\}\cup\{adj(x)\cap AL(v,\ell)\}$ and z∉C. Then, there exists a vertex y∈C such that (y,z)∉E (vertex set C is a maximal clique). Since vertex $y\in AL(v,\ell)$, there follows that vertex $z\notin adj(y)\cap AL(v,\ell)$. Thus, $\{x,y\}\cup\{adj(x)\cap adj(y)\cap AL(v,\ell)\}=C$, $1\le\ell\le L_v$. □

Observations

(1) Vertex set $\{x\}\cup B_x$ is a maximal clique in graph $G(\{x\}\cup AL(v,\ell-1))$, $x\in AL(v,\ell)$, $1\le\ell\le L_v$.

(2) Vertex set W_x has the following properties: (i) it is a subset of vertex set AL(v,ℓ), $1\le\ell\le L_v$, and (ii) if a vertex $y\in W_x$ then (y,x)∈E and $(adj(x)\cap AL(v,\ell-1))\subseteq(adj(y)\cap AL(v,\ell-1))$, i.e., $B_x\subseteq B_y$.

(3) Clique $C_{x,y}$ is a maximal clique in graph $G(\{x\}\cup W_x)$ which contains vertex $x\in AL(v,\ell)$ and $y\in W_x$, $1\le\ell\le L_v$.

Based on the previous facts and observations, we conclude that a maximal clique of the graph $G(AL(v,\ell-1)\cup AL(v,\ell))$ which contains vertices x and y, where $x\in AL(v,\ell)$ and $y\in W_x$, $1\le\ell\le L_v$, has the form

$$B_x \cup C_{x,y}$$

We are now in a position to formulate an algorithm for parallel location of all maximal cliques of a chordal graph. The algorithm, which we call MAXCLIQUES, is listed in fig. 3.

Algorithm MAXCLIQUES
input : Graph G=(V, E) and the a.l.s AL(v, 0), AL(v, 1), ..., AL(v, L_v), v∈V, of G.
output : All maximal cliques of G.
begin
1. **for** each $x \in AL(v,\ell)$, $1\le\ell\le L_v$, **do in parallel**
1.1 $B_x \leftarrow adj(x) \cap AL(v,\ell-1)$;
1.2 $F_x \leftarrow \{y \mid y \in AL(v,\ell)\ \&\ (y,z)\in E,\ \forall z \in B_x\}$;
1.3 $W_x \leftarrow \{\{x\}\cup adj(x)\}\cap F_x$;
1.4 **if** $W_x=\{x\}$ **then** $MC \leftarrow \{x\}\cup B_x$; **goto** step 2;
1.5 **for** each $y \in W_x$, **do in parallel**
$C_{x,y} \leftarrow \{x,y\}\cup\{adj(x)\cap adj(y)\}\cap W_x$;
if $C_{x,y}$ is a clique **then** $MC_x \leftarrow B_x \cup C_{x,y}$;
end;
end;
2. **for** each $x \in AL(v,\ell)$, $1\le\ell\le L_v$, **do in parallel**
for each $y \in W_x$, **do in parallel**
2.1 **if** $MC_x = MC_y$ & $x<y$ **then** $MC \leftarrow MC_x$;
end;
end;
end.

Fig. 3. The All Maximal Cliques Location Algorithm MAXCLIQUES

In substep 1.5, the algorithm computes a maximal clique $C_{x,y}$ in the graph $G(AL(v,\ell))$, which contains vertices x and y, where $x\in AL(v,\ell)$, $y\in W_x$, $1\le\ell\le L_v$. It is possible to exist two (or more) vertices $z,u\in W_x$, such that $C_{x,y}=C_{z,u}$ and $B_x=B_z$, and therefore $B_x\cup C_{x,y}=B_z\cup C_{z,u}$, i.e., $MC_x=MC_z$. In substep 2.1, we eliminate the duplicate elements of the set MC, i.e., the maximal cliques of graph $G(AL(v,\ell-1)\cup AL(v,\ell))$, $1\le\ell\le L_v$.

The correctness of the algorithm MAXCLIQUES is established through the following theorem.

Theorem 5. *Let G=(V,E) be a chordal graph. The algorithm MAXCLIQUES correctly computes the maximal cliques of the graph G.*

Proof. Let AL(v, 0), AL(v, 1), ..., AL(v, L_v) be the adjacency-level sets of the chordal graph G=(V, E), v∈V. For each vertex $x\in AL(v,\ell)$, $1\le\ell\le L_v$, the algorithm computes the maximal clique $B_x\cup C_{x,y}$, $y\in W_x$, of the graph $G(AL(v,\ell-1)\cup AL(v,\ell))$, $1\le\ell\le L_v$. Therefore, the

algorithm computes all maximal cliques of the graph $G(AL(v,\ell-1)\cup AL(v,\ell))$, $1 \le \ell \le L_v$. Since $AL(v,\ell-1) \neq AL(v,\ell+1)$, $0 < \ell < L_v$, there follows that the algorithm computes all maximal clique of the graph G=(V,E). □

5.1 The Complexity of Algorithm MAXCLIQUES

We shall obtain the overall computational complexity of the algorithm by computing the complexity of each step separately.

Step 1. This step consists of four substeps, i.e., substeps 1.1 through 1.5, which are executed sequentially. Each of these subsets is executed concurrently for n vertices. *Substep 1.1.* The intersection of two vertex sets can be executed in constant time $O(1)$ with δ processors. *Substep 1.2.* Similarly, the intersection of $|B_X|$ pairs of vertex sets is executed in constant time $O(1)$ with $|B_X|\cdot\delta \le \delta^2$ processors. *Substep 1.3.* It is easy to see that the time complexity of this substep is $O(1)$ when δ processors are available. *Step 1.4.* Clearly, the union of two vertex sets is executed in constant time $O(1)$ with δ processors. *Step 1.5.* The operation of testing whether or not a vertex set $C_{x,y}$ is a clique requires constant time $O(1)$ and δ^2 processors. Here, $|W_X|$ such a vertex sets are tested, where $|W_X| \le \delta$. Therefore, substep 1.5 can be executed in constant time $O(1)$ with δ^3 processors. Hence, the total execution time for the step 1 is constant $O(1)$, and is accomplished with $n\cdot\delta^3$ processors.

Step 2. The operation of testing whether two sets are equal or not can be executed in constant time $O(1)$ with n processors. Therefore, this step is executed in constant time $O(1)$ with $n\cdot|W_X|\cdot n \le \delta\cdot n^2$ processors.

Taking into consideration the time and processor complexity of each step of the algorithm and the fact that $\max\{n\cdot\delta^3, \delta\cdot n^2\} \le \delta^2\cdot n^2$, we can compute the overall computational complexity of the algorithm. Thus, given the adjacency-level sets of a chordal graph, the algorithm MAXCLIQUES runs in constant time $O(1)$ when $\delta^2\cdot n^2$ processors are available on a CRCW-PRAM model. Moreover, since vertex sets MC_X can be computed independently for each vertex $x \in V$, the algorithm can be also executed in time $O(\log n)$ with $\delta^2\cdot n^2/\log n$ processors. Consequently, we can formulate the following theorem.

Theorem 6. *The maximal cliques of a chordal graph G=(V,E) can be located in time* $O(\log n)$ *with* $O(\max\{\delta^2\cdot n^2/\log n, n^{\beta+D_G}\})$ *processors on a CRCW-PRAM model.*

References

1. Beame, P, Hastad, J.: Optimal Bounds for Decision Problems on the CRCW PRAM. *J. Assoc. Comput. Mach.* **36** (1989) 643-670.
2. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the Desirability of Acyclic Database Schemes, *J. Assoc. Comput. Mach.* **30** (1983) 479-513.
3. Coppersmith, A., Winograd, S.: Matrix Multiplication via Arithmetic Progression, in *STO87*, 1-6, 1987.

4. Dahlhaus, E., Karpinski, M.: Fast Parallel Computation of Perfect and Strongly Perfect Elimination Schemes, Res. Rept. RJ 5901 (59206), IBM Research Division, 1987.
5. Edenbrandt, A. Chordal Graph Recognition is in NC, *Inform. Process. Lett.* **24** (1987) 239-241.
6. Fich, F.E., Ragde , P.L., Wigderson, A.: Relations Between Concurrent-Write Models of Parallel Computation, *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing*, 179-189, 1984.
7. Fulkerson, D.R., Gross, O.A.: Incidence Matrices and Interval Graphs, *Pacific J. Math.* **15** (1965) 835-855.
8. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., New York, 1980.
9. Ho, C-W., Lee, R.C.T.: Efficient Parallel Algorithms for Maximal Cliques, Clique Tree, and Minimum Colouring on Chordal Graphs, *Inform. Process. Lett.* **28** (1988) 301-309.
10. Ho, C-W., Lee, R.C.T.: Counting Clique Trees and Computing Perfect Elimination Schemes in Parallel, *Inform. Process. Lett.* **31** (1989) 61-68.
11. Klein, P.N.: Efficient Parallel Algorithms for Chordal Graphs, *4th SIAM Conference on Discrete Mathematics*, San Francisco, CA, 1988.
12. Kucera, L.: Parallel Computation and Conflicts in Memory Access, *Inform. Process. Lett.* **14** (1982) 93-96.
13. Naor, J., Naor, M., Schaffer, A.: Fast Parallel Algorithms for Chordal graphs, *Proc. 19th ACM Symp. on Theory of Computing*, 355-364, 1987.
14. Nikolopoulos, S.D., Danielopoulos, S.D.: Parallel Computation of Perfect Elimination Schemes using Partition Techniques on Triangulated Graphs, *Computers and Mathematics with Applications* (to appear).
15. Tarjan, R.E., Yannakakis, M.: Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs, *SIAM J. Computing* **13** (1984) 566-579.
16. Vishkin, U.: Implementation of Simultaneous Memory Address Access in Model that Forbid It, *J. of Algorithms* **4** (1983) 45-50.