Efficient EREW-PRAM Parallel Colouring of Permutation Graphs

M. Andreou and S.D. Nikolopoulos

16-96

Technical Report No. 16-96/1996

Department of Computer Science
University of Ioannina
45 110 Ioannina, Greece

Efficient EREW-PRAM Parallel Colouring of Permutation Graphs

Maria Andreou

Department of Computer Engineering and Informatics,
University of Patras, GR-26500 Patras, Greece
mandreou@ceid.upatras.gr

Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina
GR-45110 Ioannina, Greece
stavros@cs.uoi.gr

Abstract — We show that the problem of colouring a permutation graph can be solved in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model of computation, where δ is the degree of the graph. Given a permutation π or its corresponding permutation graph, we construct a directed acyclic graph using certain combinatorial properties of π and, then, we compute longest paths in the directed acyclic graph. We show that the problem of colouring a permutation graph is equivalent to finding longest paths in its acyclic digraph. Our results improve in performance upon the parallel algorithm recently proposed by C-W Yu and G-H Chen [25], which solves the colouring problem in $O(\log^2 n)$ time using $n^3 / \log n$ processors on a CREW PRAM model of computation.

Keywords: Parallel algorithms, Perfect graphs, Permutation graphs, Colouring problem, Directed acyclic graphs, Longest paths.

1. Introduction

A *undirected graph* is a pair G=(V,E), where V is a finite set of n elements called *vertices* and E is a set of m unordered vertex pairs called *edges*. An undirected graph G = (V, E), with vertices numbered from 1 to n, i.e., V = {1, 2, ..., n}, is called a *permutation graph* if there exists a permutation $\pi = [\pi_1, \pi_2, ..., \pi_n]$ on $N = \{1, 2, ..., n\}$ such that,

$$(i, j) \in E \Leftrightarrow (i - j)\,(\pi^{-1}(i) - \pi^{-1}(j)) < 0$$

for all $i, j \in N$, where π_i^{-1}, denoted here as $\pi^{-1}(i)$, is the index of the element i in π [4, 6]. Given a permutation π, we shall denote, hereafter, its corresponding permutation graph as $G[\pi]$.

Many researchers have been devoted to the study of permutation graphs. They have proposed sequential and/or parallel algorithms for recognizing permutations graphs and solving combinatorial and optimization problems on them. For a sequential environment, Pnueli *et. al.* [16] gave an $O(n^3)$

time algorithm for recognizing permutation graphs using the transitive orientable graph test. Later, Spinrad [19] improved their results by designing an $O(n^2)$ time algorithm for the same problem. In the same paper, Spinrad also proposed an algorithm that determines whether or not two permutation graphs are isomorphic in $O(n^2)$ time. In [20], Spinrad *et. al.* proved that a bipartite permutation graph can be recognized in linear time by using some good algorithmic properties of such a graph. They also studied other combinatorial and optimization problems on permutation graphs. Supowit [21] solved the colouring problem, the maximum clique problem, the cliques cover problem and the maximum independent set problem, all in $O(n\log n)$ sequential time. Moreover, Farber and Keil [5] solved the weighted domination problem and the weighted independent domination problem in $O(n^3)$ time, using dynamic programming techniques. Later, Brandstadt and Kratsch [3] published an $O(n^2)$ time algorithm for the weighted independent domination problem. Atallah *et. al.* [1] solved the independent domination set problem in $O(n\log^2 n)$ time, while Tsai and Hsu [22] solved the domination problem and the weighted domination problem in $O(n\log^2 n)$ time and $O(n^2\log^2 n)$ time, respectively. Tsukiyama *et. al.* [23] proposed an algorithm that generates all maximal independent sets of a general graph in $O(nma)$ time, where a is the number of the generated maximal independent sets of the graph. Leung [12] gave algorithms for generating all maximal independent sets of interval, circular-arc and triangulated (or chordal) graphs. His algorithm takes $O(n^2+k)$ time for interval and circular-arc graphs, and $O((n+m)a)$ time for triangulated graphs, where k is the number of vertices generated. In [26], Yu and Chen showed that the generation of all the maximal independent sets can be completed in $O(n\log n+k)$ time using $O(n\log n)$ space.

Although many sequential algorithms have been proposed for permutations graphs, few parallel algorithm have been appeared in the literature. Due to work of Helmbold and Mayr [7] and Kozen *et. al.* [10], the problem of recognizing permutation graphs was shown to be in the NC class. Helmbold and Mayr presented a parallel algorithm that recognizes a permutation graph in $O(\log^3 n)$ time using $O(n^4)$ processors on a CRCW PRAM model of computation. They also solved the weighted clique problem and the colouring problem in $O(\log^3 n)$ time using $O(n^4)$ processors on same model of computation. Moreover, given a permutation graph, their algorithm can construct the permutation that represents the permutation graph.

Our objective is to study the colouring problem of permutation graphs. Recently, Yu and Chen [25] proposed a technique that transfer the colouring problem into the largest-weight path problem. Specifically, their technique, first, transforms a permutation graph (combinatorial object) into a set of planar points (geometrical object), then constructs an acyclic directed graph by exploiting the domination relation within the geometric object, and finally, solves the largest-weight path problem on the acyclic directed graph. The parallel algorithm they proposed can solve the colouring problem in $O(\log^2 n)$ time with $O(n^3/\log n)$ processors on a CREW PRAM, or in $O(\log n)$ time with $O(n^3)$ processors on a CRCW PRAM model of computation. Moreover, they proposed parallel algorithms that solve the weighted clique problem, the weighted independent set problem, the cliques cover problem, and the maximal layers problem with the same time and processor complexities.

In this paper, we present a parallel algorithm for solving the problem of colouring a permutation graph in $O(\log n \log\delta)$ time with $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model of computation, where δ is the degree of the permutation graph. Our algorithm transform the colouring problem into the longest path problem on acyclic directed graphs. Specifically, given a permutation π on $N = \{1, 2, ..., n\}$, the algorithm directly constructs an acyclic

directed graph $G^*[\pi] = (V^*, E^*)$ having $V^* = V \cup \{s\}$, and then, computes the length of the longest paths from a specific vertex $s \in V^*$ to every vertex $v \in V^*$. We prove that there is an one-to-one correspondence between the length of the longest path from s to a vertex $v \in V^*$ and the colour of v in $G[\pi]$.

The paper is organised as follows. In Section 2, we establish the notation and terminology we shall use throughout the paper. In Section 3, we describe the method that transform a given permutation π into an acyclic directed graph, and we prove that colouring the permutation graph $G[\pi]$ is equivalent to finding the length of longest paths on its acyclic directed graph. In Section 4, we propose an EREW PRAM parallel algorithm for solving the longest path problem on acyclic directed graphs. In Section 5, we address some comparison issues of both our algorithm and Yu and Chen's algorithm.

2. Definitions

A *colouring* of a graph is an assignment of colours to its vertices so that no two adjacent vertices have the same colour. The set of all vertices with any one colour is independent and is called a *colour class*. The *colouring problem* is to colour a graph with as less as possible colours (minimum number of colours).

Permutations may be represented in many ways. The most straightforward is simply a rearrangement of the numbers 1 through n, as in the following example where $n = 9$.

index	1	2	3	4	5	6	7	8	9
permutation	8	3	2	7	1	9	6	5	4

Suppose π is a permutation on $N = \{1, 2, ..., n\}$. Let us think of π as the sequence $[\pi_1, \pi_2, ..., \pi_n]$, so, for example, the permutation $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$ has $\pi_1 = 8$, $\pi_2 = 3$, etc. Notice that $(\pi^{-1})_i$, denoted here as $\pi^{-1}(i)$, is the position of element i in the sequence π. In our example $\pi^{-1}(8) = 1$, $\pi^{-1}(3) = 2$, etc.

Let π be a permutation on $N = \{1, 2, ..., n\}$. An element i is said to be *dominated* by the element j (or j *dominates* i) if $i < j$ and $\pi^{-1}(i) > \pi^{-1}(j)$. An element i is said to be *directly dominated* by the element j (or j *directly dominates* i) if i is *dominated* by j and there exists no element k in π such that i is dominated by k and k is dominated by j. In the permutation given above, 1, 6, 5, 4 are dominated by 7 and 1, 6 are directly dominated by 7. We shall use, hereafter, the notation *D-dominates* and *D-dominated* for the terms *directly dominates* and *directly dominated*, respectively. The *domination set* (*D-domination set*) of an element i of the permutation π is the set which contains all the elements of π that dominate (*D*-dominate) i.

Given a permutation π on $N = \{1, 2, ..., n\}$, its corresponding permutation graph $G[\pi] = (V, E)$ can be constructed as follows: $G[\pi]$ has vertices numbered from 1 to n; two vertices are jointed by an edge if the larger of their corresponding numbers is to the left of the smaller in permutation π, i.e., $G[\pi]$ has n vertices $v_1, v_2, ..., v_n$, and m edges such that $(i, j) \in E$ iff $(i - j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$.

Let $G[\pi] = (V, E)$ be a permutation graph. A vertex $v_j \in V$ is said to be a neighbour (D-neighbour) of vertex v_i if j dominates (D-dominates) i in the permutation π. For example, in the graph $G[\pi]$ vertices v_8, v_7, v_9, v_6 are neighbours of v_5, while vertex v_6 is D-neighbour of v_5.

We conclude this section with some graph-theoretic notation employed in this paper. A *path* P in a graph $G = (V, E)$ is a sequence of vertices $[v_0, v_1, ..., v_k]$ such that $(v_{i-1}, v_i) \in E$, $i = 1, 2, ..., k$; P is a path from v_0 to v_k of *length* k; P is directed or undirected on whether G is directed or undirected graph. The path P is a *simple path* if $v_0, ..., v_{k-1}$ are distinct and $v_1, ..., v_k$ are distinct, and all edges on P are distinct. A simple path $[v_0, v_1, ..., v_k]$ is a *cycle* if $v_0 = v_k$; otherwise it is *noncyclic*. A *directed acyclic graph* (dag or DAG) is a digraph with no cycles.

3. Problem Transformation and Solution

We have referred to the problem of colouring a graph as one of trying to assign particular colours to its vertices so that no two adjacent vertices have the same colour. Moreover, the number of colours used must be as less as possible. The key to the solution is to find the colour classes of the graph, i.e., the classes of vertices that can be coloured with the same colour. To this end, one can think of transforming the graph into another combinatorial object (e.g., tree, directed graph, etc.) and, then, solving a particular problem on this object (e.g., vertices lying in the same level of the tree, vertices having the same distance from a particular vertex, etc.) which gives the solution to the colouring problem.

In this work, we use a strategy, similar to that used in [25], to transform a permutation graph $G[\pi]$ into an acyclic directed graph $G^*[\pi]$. Then, we solve the colouring problem on $G[\pi]$ by solving the longest path problem on $G^*[\pi]$. In particular, given a permutation π (or its corresponding graph), we construct the acyclic directed graph $G^*[\pi]$ by exploiting the D-domination relation, as follows:

(i) for every element i in permutation π, add the vertex v_i to $G^*[\pi]$, i.e., $v_i \in V^*$;
(ii) compute the D-domination set of each element i in permutation π;
(iii) if k is D-dominated by i then add the edge $\langle v_i, v_k \rangle$ to $G^*[\pi]$, i.e., $\langle v_i, v_k \rangle \in E^*$;
(iv) add a dummy vertex s in V^* and set $\langle s, v_i \rangle \in E^*$ for every v_i with indegree$(v_i) = 0$;

Figure 1 shows a permutation $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$ and its corresponding permutation graph $G[\pi]$, while Figure 2 shows the results of the transformation of the permutation graph $G[\pi]$ into a directed acyclic digraph using the transformation strategy described above. Obviously, the resulting graph is the directed acyclic digraph $G^*[\pi]$.

index	1	2	3	4	5	6	7	8	9
permutation	8	3	2	7	1	9	6	5	4

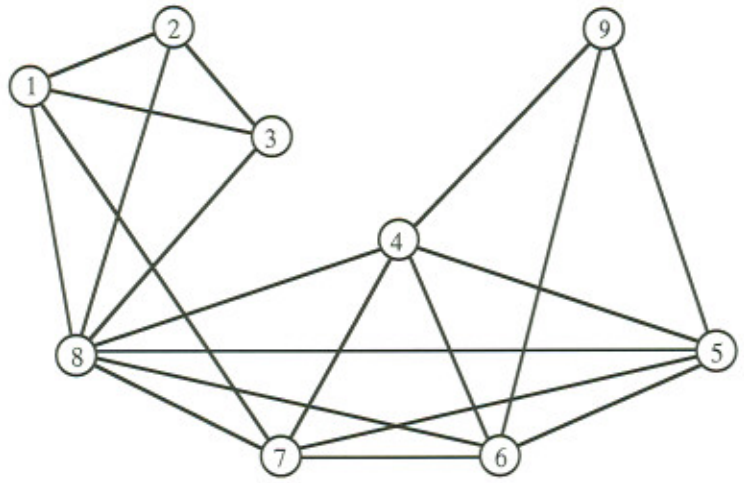

Fig. 1: A permutation $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$, and its corresponding permutation graph $G[\pi]$.

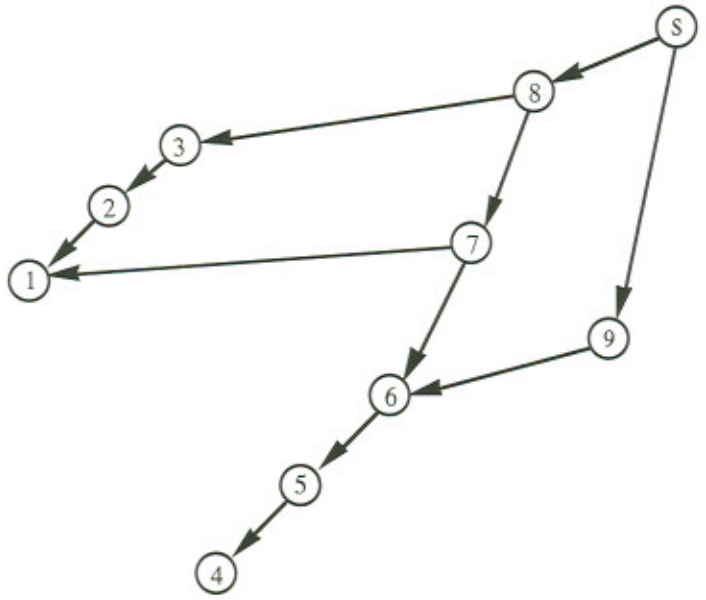

Fig. 2: The acyclic directed graph $G^*[\pi]$ obtained from permutation π.

We prove that there is an one-to-one correspondence between the length of the longest path from s to a vertex v in $G^*[\pi]$ and the colour of vertex v in $G[\pi]$. To this end, we include here the results of Yu and Chen [25].

Lemma 3.1. There exist a path from v_i to v_j in $G^*[\pi]$ iff i dominates j (j is dominated by i; or (v_i, v_j) is an edge in $G[\pi]$).

Lemma 3.2. There exist a path $[v_i, v_j, ..., v_k]$ in $G^*[\pi]$ iff $[i, j, ..., k]$ is a decreasing subsequence in the permutation π (the subgraph of $G[\pi]$ induced by $\{v_i, v_j, ..., v_k\}$ is a clique).

Lemma 3.3. Let $lp(v_i)$ and $lp(v_j)$ be the lengths of the longest paths $[s, ..., v_i]$ and $[s, ..., v_j]$ in $G^*[\pi]$. If (v_i, v_j) is an edge in $G[\pi]$, then $lp(v_i) \neq lp(v_j)$.

Having shown the relation between the colouring problem on a permutation graph $G[\pi]$ and the longest path problem on $G^*[\pi]$, we are in a position to formulate a parallel algorithm for solving the colouring problem on permutation graphs. The idea of the algorithm is motivated by the work performed by Yu and Chen [25] and the combinatorial properties of permutations. It consists of three steps:

Algorithm `Colouring`:
Input : A permutation π and its corresponding graph $G[\pi] = (V, E)$.
Output : The colour of each vertex $v_i \in V$, $i = 1, 2, ..., n$.
begin
1. Transform the permutation graph $G[\pi] = (V, E)$ into a dag graph $G^*[\pi] = (V^*, E^*)$ as follows:
   1.1. $V^* \leftarrow V$;
   1.2. Edge $< v_k, v_i > \in E^*$ iff v_k is a *D-neighbour* of v_i (*k D-dominates i* or *i* is *D-dominated* by *k*).
   1.3. Add a dummy vertex s in V^* and set: $< s, v_i > \in E^*$ for every v_i with indegree(v_i) = 0;
2. For each vertex $v_i \in V^* - \{s\}$ compute the length $lp(v_i)$ of the longest path from s to v_i;
3. Set $colour(v_i) \leftarrow lp(v_i)$, $i = 1, 2, ..., n$.
end;

In step 3, the algorithm colours the vertices of graph G with k colours, where k is the maximum of the lengths of all the longest paths in digraph $G^*[\pi]$, $k \leq n$. Vertices v_i and v_j are coloured with the same colour if the longest paths from s to v_i and v_j, respectively, have the same length. The correctness of the algorithm is established through the Theorem 3.1. Its proof relies on the results of [25] and Lemmas 3.1 through 3.3.

Theorem 3.1. Given a permutation π, the algorithm `Colouring` correctly solves the colouring problem on its corresponding permutation graph.

4. The Main Results

Having defined the directed acyclic graph $G^*[\pi] = (V^*, E^*)$ of a given permutation π, and showed the one-to-one correspondence between the colouring problem on $G[\pi]$ and the longest path

problem on $G^*[\pi]$, we are in a position to formulate a parallel algorithm for computing the lengths of the longest paths from vertex $s \in V^*$ to every vertex $v \in V^*$.

4.1 Construction of the directed acyclic digraph $G^*[\pi]$

As we saw previously, the directed acyclic graph $G^*[\pi]$ is constructed by exploiting the *D-domination* relation on permutation π. Therefore, there is a need of computing the *D*-domination set for every element in π. Obviously, the *D*-domination set of an element is subset of its domination set. So, we first compute the domination set for every element i in permutation π, and then select from it the elements that constitute the *D*-domination set.

We can easily show that, the domination set of an element, say i, in a permutation π, is simply the set which contains all the elements that are greater than i and laying on the left of the element i in π (see the definition of the domination set in Section 2). Towards the computation of the domination set of the element i, we define two arrays of length n and δ, respectively, where δ is the degree of the permutation graph $G[\pi]$.

D'_i array of length n; it contains the elements of π that dominate i;
That is, j is an element of D'_i if $i < j$ and $\pi^{-1}(i) > \pi^{-1}(j)$. It is obvious that the elements that dominate i are at most δ.

D_i array of length δ; it contains the elements of D'_i in consecutive positions;

Notice that the degree δ of the permutation graph $G[\pi]$ can be computed as follows:

$$\delta = \max_{1 \le i \le n} \sum_{j=1}^{n} D''_i[j]$$

there $D''_i[j] = 1$ if $D'_i[j] > 0$, $i = 1, 2, ..., n$.

The computation of each D_i, $1 \le i \le n$, can be done independently, and therefore in parallel. The following parallel algorithm describes this computation.

Algorithm `Domination`:
Input : A permutation π on $\{1, 2, ..., n\}$;
Output : The set D_i of all the elements which dominate i, $1 \le i \le n$;
begin
1. For every i, $1 \le i \le n$, do in parallel
   for every j, $1 \le j \le n$, do in parallel
   if $i < j$ and $\pi^{-1}(i) > \pi^{-1}(j)$ then $D'_i[\pi^{-1}(j)] \leftarrow j$;
2. For every i, $1 \le i \le n$, do in parallel
   Store all the non-zero elements from array D'_i into consecutive memory locations in D_i, such that element x is on the left of y in D_i iff x is on the left of y in D'_i, for every pair of elements x, y in D_i.

end;

Having computed the domination set of each element i of π, let us now describe the computation of the *D*-domination set of the element i, $1 \le i \le n$.

Based on the definition of the D-domination relation of two elements of a permutation π, we can easily show that, the last element of the array D_i, say j, directly dominates the elements i. Therefore, the D-domination set contains the element j and every other element z of the array D_i, such that $z < j$ and there is no element z' smaller than both z and j, i.e. $z' < z$, and $z' < j$, and $\pi^{-1}(z) < \pi^{-1}(z') < \pi^{-1}(j)$.

Next, we list an algorithm that computes the array DD_i of length δ, which contains all the elements of D_i that directly dominate i, $1 \le i \le n$.

Algorithm `D-domination`:

Input : The set D_i of all the elements which dominate i, $1 \le i \le n$;

Output : The set DD_i of all the elements which D-dominate i, $1 \le i \le n$;

begin

1. Set $Y_i \leftarrow D_i$, $1 \le i \le n$;
2. If Y_i is in increasing order, then go to step 6;
3. Compute the maximal increasing contiguous subsequences $Y_{i,1}, Y_{i,2}, ..., Y_{i,k}$ in Y_i.
   That is, $Y_{i,1}, Y_{i,2}, ..., Y_{i,k}$ have the following properties:
   (a) $Y_{i,j}$ is in increasing order, $j = 1, 2, ..., k$;
   (b) The last element $last(i, j)$ of $Y_{i,j}$ is greater than the first element $first(i, j+1)$ of $Y_{i,j+1}$;
4. For every $Y_{i,j}$, $j = 1, 2, ..., k-1$, do in parallel
   for every element y of $Y_{i,j}$ do in parallel
   if $y > first(i, j+1)$ then $y \leftarrow first(i, j+1)$;
5. Repeat steps 2, 3 and 4;
6. Set $DD_i \leftarrow Y_i$, $1 \le i \le n$;
7. For every i, $1 \le i \le n$, do in parallel
   for every j, $1 \le j \le \delta - 1$, do in parallel
   if $Y_i[j] = Y_i[j+1]$ then $DD_i[j] \leftarrow 0$;

end;

Let us now show step-by-step the computation of the D-domination set of the element 1 of the permutation $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$, using the algorithm `D-domination`. This computation goes as follows:

- $Y_1 \leftarrow D_1 = [8, 3, 2, 7]$;
  Step 2. goto step 3;
  Step 3. $Y_{1,1} = [8]$, $Y_{1,2} = [3]$, $Y_{1,3} = [2, 7]$;
  Step 4. $Y_{1,1} = [3]$, $Y_{1,2} = [2]$, $Y_{1,3} = [2, 7]$;

- $Y_1 \leftarrow [3, 2, 2, 7]$;
  Step 2. goto step 3;
  Step 3. $Y_{1,1} = [3]$, $Y_{1,2} = [2, 2, 7]$;
  Step 4. $Y_{1,1} = [2]$, $Y_{1,2} = [2, 2, 7]$;

- $Y_1 \leftarrow [2, 2, 2, 7]$;
  Step 2. goto step 6;
  Step 6. $DD_1 = [2, 2, 2, 7]$;
  Step 7. $DD_1 = [0, 0, 2, 7]$;

The correctness of the algorithm is based on the previous discussion and is established through the following Lemma.

Lemma 4.1. Algorithm `D-domination` correctly computes the D-domination set for each element of a permutation π on $\{1, 2, ..., n\}$.

Each of the algorithms `Domination` and `D-domination` actually computes an $n \times \delta+1$ matrix. We shall refer to theses matrices as *domination matrix D* and *D-domination matrix DD*, respectively. Figure 3 shows the matrices D and DD computed by the algorithms `Domination` and `D-domination`, respectively, for the permutation π. We denote by s the element $n+1$; we have seen that s D-dominates the elements that have zero entries in the inversion table (see Section 2).

D :

	1	2	3	4	5	6	7
1	8	3	2	7			
2	8	3					
3	8						
4	8	7	9	6	5		
5	8	7	9	6			
6	8	7	9				
7	8						
8							
9							

DD :

	1	2	3	4	5	6	7
1			2	7			
2		3					
3	8						
4					5		
5				6			
6		7	9				
7	8						
8							
9							

Fig. 3: The matrices D and DD of $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$.

We have already shown the way that we can construct the directed acyclic graph $G^*[\pi]$ from a permutation π through the D-domination relation. Next, we list the parallel algorithm for constructing the directed graph $G^*[\pi]$.

Algorithm `D-domination-to-dag`:

Input : The set DD_i of all the points which D-dominates i, $1 \le i \le n$;

Output : The directed acyclic graph $G^*[\pi]$;

begin

1. Construct the dag $G^*[\pi] = (V^*, E^*)$, with vertex set $V^* = \{v_1, v_2, ..., v_n\}$, as follows:
   For every $i = 1, 2, ..., n$, do in parallel
   for every $j = 1, 2, ..., \delta$, do in parallel
   if $DD_i[j] = k > 0$ then add an arc from vertex v_k to vertex v_i, i.e., $\langle v_k, v_i \rangle \in E^*$;
2. Add a vertex s in $G^* = (V^*, E^*)$ and arcs $\langle s, v_i \rangle \in E^*$ for every vertex v_i with indegree$(v_i) = 0$;
   That is, vertex s is now the only vertex in G^* with indegree equals 0;

end;

4.2 Computation of the longest paths in G[π]*

In this section, we developed a parallel algorithm that computes longest paths in an acyclic directed

graph. In particular, given an acyclic directed graph having only one node, say s, with indegree zero, the algorithm computes the longest paths from s to every other node of the graph.

We first give the definition of the k-th neighbour of a vertex v_i of the acyclic directed graph $G^*[\pi] = (V^*, E^*)$, and then we define two arrays containing important information for the process of computation longest paths in $G^*[\pi]$.

Definition 4.2.1. Let v_i, v_j be two vertices of $G^*[\pi]$. Vertex v_j is the k-th neighbour of the vertex v_i if there are k-1 elements z in π such that $z > i$, $\pi^{-1}(z) < \pi^{-1}(i)$ and $\pi^{-1}(z) < \pi^{-1}(j)$.

It is easy to see that if v_j is the k-th neighbour of the vertex v_i then $D[i, k] = v_j$. Figure 4 depicts the relation between the vertex $v_i \in V^*$ and all the other vertices of $G^*[\pi]$. Area A contains all the neighbours of v_i, i.e., all the vertices v_z for which there exist a path from v_z to v_i and z dominates i, area D contains all the vertices v_z for which there exist a path from v_i to v_z, while areas B and C contain all the vertices v_z for which there exist no path from v_i to v_z. Vertex v_p is the j-th neighbour of vertex v_i, while v_q is the k-th neighbour of vertex v_i.

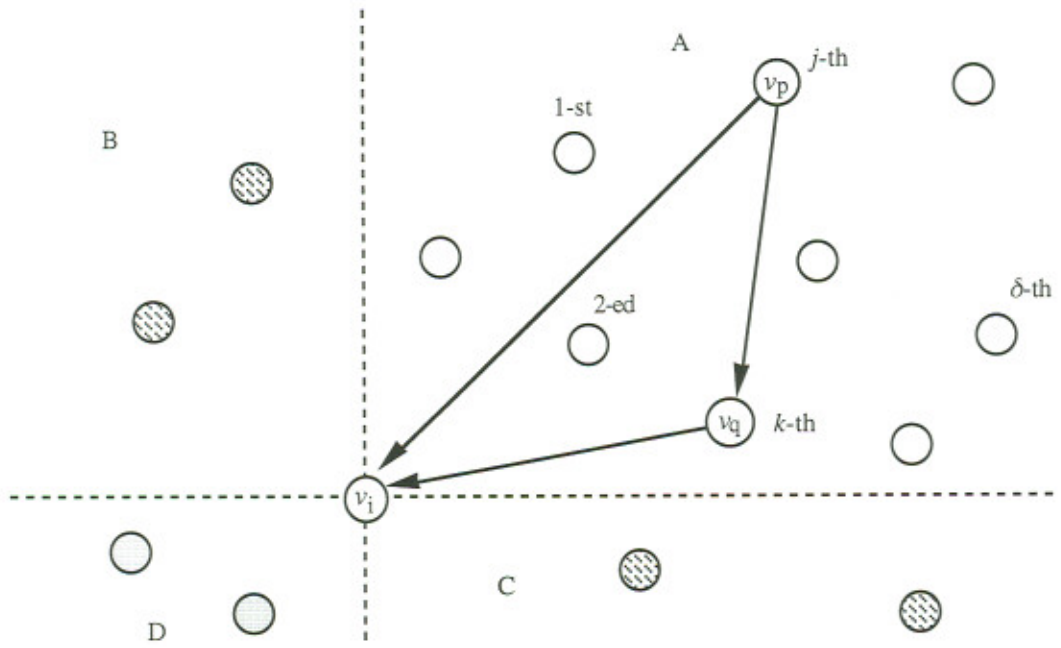

Fig. 4: The relation between the vertex $v_i \in V^*$ and all the other vertices of $G^*[\pi]$.

Now, for each node $v \in V$ we define two arrays of length δ+1 and $\delta \times \delta$+1, respectively, where δ is the degree of the permutation graph $G[\pi]$.

L_i array of length δ+1; it contains for each neighbour v_j of vertex v_i (j dominates i) the length of a path (not always the longest) from v_j to v_i; Obviously, the elements that dominate i are at most δ+1 (at most δ elements plus the element s);

LL_i array of length $\delta \times \delta$+1; it contains the information of all the L_j where v_j is a neighbour of vertex v_i. Specifically, $LL_i[j, k]$ contains the length of a path (if such a path exists) from the j-th neighbour of v_i to the k-th neighbours of v_i, if $j < k$ and $D_i[j] > D_i[k]$;

We are now in a position to formulate a parallel algorithm for solving the longest path problem on the acyclic digraph $G^*[\pi]$. It consists of four steps:

Algorithm `Longest-Paths`:

Input : The directed acyclic graph $G^*[\pi]$;

Output : The lengths of the longest paths from s to every vertex v_i in $G^*[\pi]$, $1 \le i \le n$;

begin

1. For every vertex v_i of G^*, $1 \le i \le n$, do in parallel
   (a) initialise the array $L_i[1..\delta+1]$ as follows:
   Set $L_i[j] = 1$ if the j-th neighbour of v_i is a D-neighbour of v_i,
   i.e., $L_i[j] \leftarrow 1$ if $\langle v_i, v_{D[i,j]}\rangle \in E^*$ or equivalently if $v_{D[i,j]} \in DD_i$;
   (b) initialise the matrix $LL_i[\delta..\delta+1]$ as follows:
   Set $LL_i[j, k] = 1$ if the j-th neighbour of v_i is a D-neighbour of v_i and the k-th neighbour of v_i is a D-neighbour of v_j,
   i.e., $LL_i[j, k] \leftarrow 1$, if $\langle v_{D[i,k]}, v_{D[i,j]}\rangle \in E^*$or equivalently if $v_{D[i,k]} \in DD_{D[i,j]}$;
2. For every vertex v_i of G^* do in parallel
   2.1 Let x, y be the j-th and the k-th neighbours of v_i, respectively,
   i.e., $x = D[i, j]$ and $y = D[i, k]$;
   If y is a neighbour of x, i.e., if $y \in D_x$, then returns the column clm of D_x to which y belongs, i.e., if $D[x, z] = y$ then $clm \leftarrow z$;
   Set $LL_i[j, k] \leftarrow L_j[clm] + L_i[k]$;
   2.2 Compute the maximum element $max_i[k]$ of each column of $LL_i[1..\delta, k]$, $k = 1, 2, ..., \delta+1$;
   2.3 If $max_i[k] > L_i[k]$ then $L_i[k] \leftarrow max_i[k]$;
3. Repeat $\log n$ time the steps 2;
4. Set $lp(v_i) \leftarrow L_i[s]$, $i = 1, 2, ..., n$.

end;

Theorem 4.1. Given a permutation π and the directed acyclic graph $G^*[\pi]$, the algorithm `Longest-Paths` correctly computes the length of the longest paths from vertex s to every other vertex of the graph $G^*[\pi]$.

In order to illustrate the workings of algorithm `Longest-Paths`, we present with a help of an example the processes of two consecutive steps. Bellow each array L_i, $1 \le i \le n$, we have written the i-th row of the array D_i (see Fig. 3).

Figure 5 shows the arrays L_i of all the vertices v_i which are neighbours of vertex v_4 and the matrix LL_4 after the execution of the ($\log n$-1)-th step of the algorithm, while Figure 6 shows the arrays L and LL of vertex v_4 after the execution of the $\log n$-th step of the algorithm. Since the $\log n$-th step is the last step of the algorithm, the position s of array L_4 obviously contains the solution, i.e., it contains the length of the longest path from s to v_4 and, thus, the colour of vertex v_4. In the permutation graph $G[\pi]$, where $\pi = [8, 3, 2, 7, 1, 9, 6, 5, 4]$, the colour of vertex v_4 is 5.

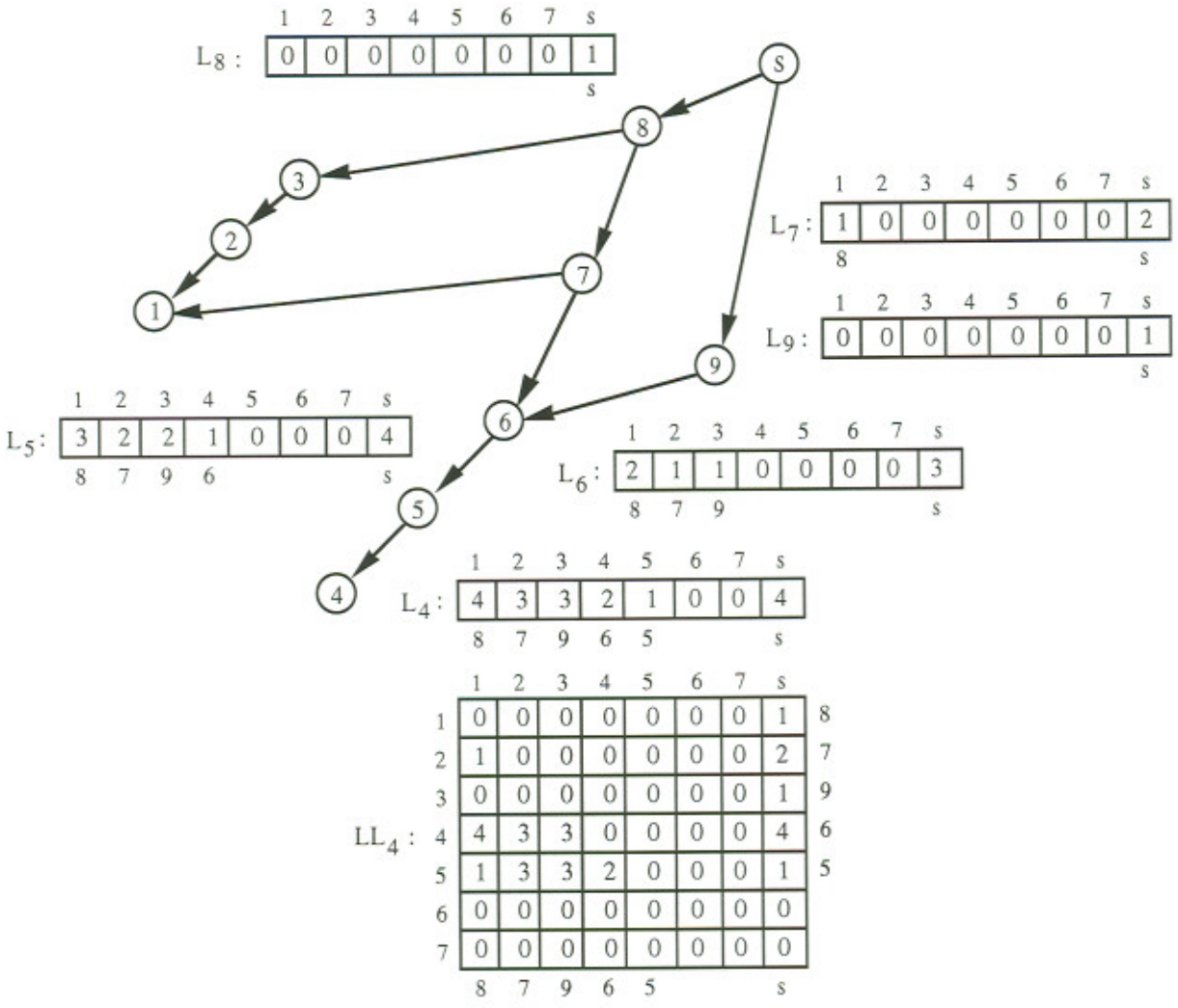

Fig. 5: The results after execution of the step log$n$ - 1 of algorithm `Longest-Paths`.

5. Complexity of the Algorithms

The model of parallel computation used in this paper is the well-known Exclusive-Read, Exclusive-Write PRAM model (EREW PRAM) [9, 17]. In this model, the operations of union ($\cup$), intersection ($\cap$) and subtraction (-) are executed in $O(\log n)$ time by $O(n / \log n)$ processors. The maximum and/or the minimum of n elements can also be computed within the same time and processor complexity. Moreover, in this model, an element can be copied n times in the shared-memory in $O(\log n)$ time using $O(n / \log n)$ processors (see [9]).

Let us now analyse the computational complexity of all the parallel algorithms we developed using the EREW PRAM model. For all cases, we shall obtain the overall complexity by computing the complexity of each step separately.

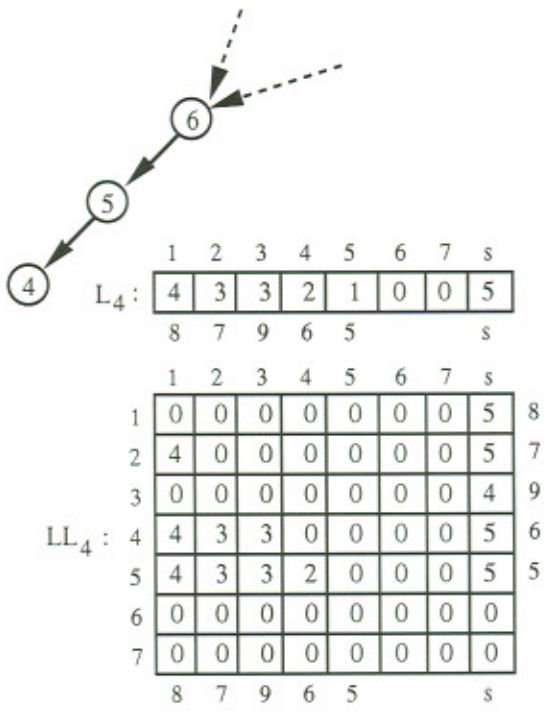

Fig. 6: The results after execution of the step log$n$ of algorithm `Longest-Paths`.

5.1 The Domination algorithm

In order to execute algorithm *Domination* on the EWER PRAM model, there is a need of copying in the shared-memory n times the n elements of π. This computation can be done in $O(\log n)$ using n^2 / logn processors on the EREW model of computation. Therefore, the complexity of each step of the algorithm can be computed as follows:

Step 1. Obviously, this step can be executed in $O(\log n)$ time using $n^2 / \log n$ processors. *Step 2*. The problem of storing all the non-zero elements of an array of length n into consecutive memory locations is equivalent to the problem of computing the prefix-sums of its n elements. It is well-known that the prefix-sums of n elements can be computed in $O(\log n)$ time using $n / \log n$ processors. Thus, the following theorem holds.

Theorem 5.1. Given a permutation π, the dominated set of each element of π can be computed in $O(\log n)$ time using $n^2 / \log n$ processors on the EREW PRAM model of computation.

5.2 The D-domination algorithm

In order to use the EREW PRAM model, we have to produce δ copies of the $n \times \delta$ domination matrix D, where δ is the maximum decreasing sequence in permutation π (n arrays D_i of length δ). This operation can be executed in $O(\log \delta)$ with $\delta^2 n / \log \delta$ processors. Let us now analyse the computational complexity of the algorithm D-domination.

The algorithm consists of 7 steps: *Step 1*. The assignment operation on δn elements takes $O(1)$ time by using δn processors or $O(\log\delta)$ time by using $\delta n / \log\delta$ processors. *Step 2*. The operation of testing whether a sequence of δ elements is in increasing order or not can be executed in $O(\log\delta)$ time with $\delta / \log\delta$ processors. Therefore, this steps is executed in $O(\log\delta)$ time with $\delta n / \log\delta$ processors. *Step 3*. This step computes the maximal increasing contiguous subsequences of a sequence of length δ. This operation is executed in $O(\log\delta)$ time using $\delta / \log\delta$ processors (here, we have again the problem of storing all the non-zero elements of an array of length δ into consecutive memory locations; see step 2 of algorithm Domination). Thus, step 3 can be executed in $O(\log\delta)$ time with $\delta n / \log\delta$ processors. *Step 4*. In this step some comparison and assignment operations are done in each subsequence computed in the previous step. There are at most δn such subsequences, and thus, this step can be executed in $O(\log\delta)$ time with $\delta n / \log\delta$ processors. *Step 5*. Here, the steps 2, 3 and 4 are repeated $O(\log\delta)$ time. That is, the time complexities of the steps 2, 3 and 4 are increased by the factor $O(\log\delta)$, and thus, each of them has time complexity $O(\log^2\delta)$. *Step 6*. This step has the same time and processor complexity as step 1. *Step 7*. Obviously, this step can be executed in $O(\log\delta)$ time with $\delta n / \log\delta$ processors.

Take into consideration the time and processor complexity of each step of the algorithm, and the complexity of copying the domination matrix D, we conclude that it can be executed in $O(\log^2\delta)$ time using $\delta^2 n / \log\delta$ processors, and, thus, we have the following lemma.

Lemma 5.1. Given the domination matrix D of a permutation π, the algorithm `D-domination` computes the direct dominated set of each element of π in $O(\log^2\delta)$ time using $\delta^2 n / \log\delta$ processors on the EREW PRAM model of computation.

The algorithm `D-domination` takes as input the domination matrix D, which computed by algorithm `Domination` in $O(\log n)$ time using $n^2 / \log n$ processors (see Theorem 5.1). Thus, we obtain the following theorem.

Theorem 5.2. Given a permutation π and the dominated set of each element of π, the direct dominated set of each element of π can be computed in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model of computation.

5.3 The D-domination-to-DAG algorithm

We can easily compute the time and processor complexity of the algorithm `D-domination-to-DAG` on the EREW PRAM model.

The algorithm consists of two steps. *Step 1*. It can be executed in $O(\log\delta)$ time using $\delta n / \log\delta$ processors. *Step 2*. To find a vertex with indegree zero in graph $G^*[\pi]$ is equivalent to find a row D_i of the $n \times \delta$ matrix D having all its entries 0. Therefore, this step can be executed in $O(\log\delta)$ time using $\delta n / \log\delta$ processors.

From the above step-by-step analysis, we obtain that the algorithm `D-domination-to-DAG` is executed in $O(\log\delta)$ time using $\delta n / \log\delta$ processors on the EREW PRAM model. So, the following lemma is hold.

Lemma 5.2. Given the D-domination matrix DD of a permutation π, the algorithm `D-domination-to-DAG` constructs the directed acyclic graph $G^*[\pi]$ in $O(\log\delta)$ time using $\delta n / \log\delta$ processors on the EREW PRAM model of computation.

The algorithm `D-domination-to-DAG` takes as input the D-domination matrix DD, which is computed by algorithm `D-domination` in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors, when the permutation π is given. Therefore, we have the following result.

Theorem 5.3. Given a permutation π, the directed acyclic graph $G^*[\pi]$ is constructed in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model of computation.

5.4 The Longest-Paths algorithm

Let us now compute the complexity of algorithm `Longest-Paths`, which obviously is the main algorithm of this work. For EREW execution of this algorithm, we have to produce δ copies of the $n\times\delta$ domination matrix D, and δ copies of each L_i, $i = 1, 2, ..., n$, i.e., δ copies of δn elements. We have shown that both operations can be executed in $O(\log\delta)$ with $\delta^2 n / \log\delta$ processors on the EREW PRAM model.

Using the EREW parallel model of computation, the time and processor complexity of the algorithm is analyzed as follows:

Step 1. The first step of the algorithm initializes the array L_i of length $\delta+1$ and the matrix LL_i of length $\delta \times \delta+1$, for $1 \le i \le n$. These operations are executed in $O(\log\delta)$ time by using $\delta^2 n / \log\delta$ processors. *Step 2*. This step consists of substeps 2.1, 2.2 and 2.3, which are executed for every vertex v_i of $G^*[\pi]$, $1 \le i \le n$. *Substep 2.1*. All the operations in this substep can be done in $O(\log\delta)$ time with $\delta^2 / \log\delta$ processors, by using the domination matrix D. *Substep 2.2*. Here, the maximum element of each column of the $\delta \times \delta+1$ matrix LL_i are computed. This operation can be executed in $O(\log\delta)$ time with $\delta(\delta+1) / \log\delta$ processors. *Substep 2.3*. In this substep, $O(\delta)$ comparisons are made, which, obviously, can be done in $O(\log\delta)$ time with $\delta / \log\delta$ processors. Take in to consideration the complexity of substeps 2.1, 2.2 and 2.3, we conclude that the overall time complexity of step 2 and number of processors used in this step, are $O(\log\delta)$ and $\delta^2 n / \log\delta$ respectively. *Step 3*. This step causes the execution $\log n$ times of the step 2. Thus, step 3 are executed in $O(\log n \log\delta)$ time using $\delta^2 n / \log\delta$ processors. *Step 4*. In the step of the algorithm, n assignments are made. We can easily see that, this step is executed in $O(\log n)$ time with $n / \log\delta$ processors.

Take into consideration the time and processor of each step of the algorithm, we can formulate the following theorem.

Theorem 5.4. Given the directed acyclic graph $G^*[\pi]$ having vertex s with indegree(s) = 0, the length of the longest paths from s to every other vertex of the graph can be computed in $O(\log n \log\delta)$ time using $\delta^2 n / \log\delta$ processors on the EREW PRAM model of computation.

5.5 The Colouring algorithm

By design, the algorithm `Colouring` incorporate all the operation described in the previous algorithms. In particular, all the operations of step 1 are executed in $O(\log n \log\delta)$ time with $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors using the algorithm `D-domination-to-DAG` (see Theorem 5.2), while all the operations of step 2 are executed in $O(\log n \log\delta)$ time with $\delta^2 n / \log\delta$ processors using the algorithm `Longest-Paths` (see Theorem 5.4). Obviously, step 3 is executed in $O(\log n)$ time using $n / \log n$ processors on the EREW PRAM model.

Take into consideration the time and processor of the algorithms `D-domination-to-DAG` and `Longest-Paths`, and the time and processor of the step 3, we present the following result.

Theorem 5.5. The problem of colouring permutation graphs can be solved in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model of computation.

6. Conclusions

In this paper we study the problem of colouring permutations graphs. We propose an efficient parallel algorithm which colours a permutation graph in $O(\log n \log\delta)$ time using $max\{n^2 / \log n, \delta^2 n / \log\delta\}$ processors on the EREW PRAM model, where n and δ are the number of vertices and the degree of the permutation graph, respectively.

The idea of our algorithm is motivated by the work performed by C-W Yu and G-H Chen [25]. They presented an algorithm which takes as input a permutation graph and transforms it into a set of planar points, constructs an acyclic directed graph, and finally solves the largest-weight path problem on this acyclic digraph. Using this strategy, they solve the colouring problem on permutation graph in $O(\log^2 n)$ time using $n^3 / \log n$ processors on a CREW PRAM model of computation or in $O(\log n)$ time using n^3 processors on a CRCW PRAM [2, 11, 24]. On the other hand, we designed an algorithm which takes as input a permutation π, and directly constructs an acyclic directed graph using combinatorial properties on π. Then, it solves the colouring problem on the permutation graph $G[\pi]$ by solving the longest path problem on the constructed acyclic digraph.

Let us now comment on the most important design issues of both algorithms. First, we focus on an important figure of the algorithms which is the computational model they uses. Specifically, our algorithm can be executed on the EREW PRAM model, while Yu and Chen's algorithm is executed on a CREW PRAM. Therefor, our algorithm is more efficient since, as it is well-known, the former model is less powerful than the later one. The second comment concerns the time and processor complexity of the colouring algorithms. In all the cases, exempt that the permutation graph is complete, our algorithm outperforms the algorithm proposed by Yu and Chen. Moreover, in the case where $\delta \leq \sqrt{n}$, our algorithm can be executed using only $n^2 / \log n$ processors on the EREW PRAM. Another point we should stress concerns the transformation strategy. Our algorithm avoids to transform the given permutation or its corresponding permutation graph into a set of planar points. Instead, it constructs the acyclic directed graph using certain combinatorial properties of permutation π. This construction process has the following effect: the two acyclic directed graphs constructed by the two algorithms are not the same. Specifically, in our case the edge set of the

acyclic digraph has much less elements than the edge set of the permutation graph has. (Notice that, the number of edges in the Yo and Chen's acyclic digraph is equal to the number of edges of permutation graph.) That is, the parameter δ which affects the performance of our algorithms is not actually the degree of the permutation graph; it takes less values than the degree of the permutation graph.

In closing, we point out that we are now looking into the colouring problem using the Lattice representation of a permutation [18] and the relationship between permutations and binary search trees. Moreover, we are studying the colouring problem for some other classes of perfect graphs [6, 8, 13, 14, 15] using CREW or CRCW PRAM's.

References

[1] M.J. Atallah, G.K. Manacher and J. Urrutia, Finding a minimum independent dominating set in a permutation graph, *Discrete Applied Mathematics*, vol. 21, pp. 177-183, 1988.

[2] P. Beame and J. Hastad, Optimal bounds for decision problems on the CRCW PRAM, *J. Assoc. Comput. Mach.*, vol. 36. pp. 643-670, 1989.

[3] A. Brandstadt and D. Kratsch, On domination problems for permutation and other graphs, *Theoretical Computer Science*, vol. 54, pp. 181-198, 1987.

[4] L. Chen, Logarithmic time NC algorithms for comparability graphs and circle graphs, *Lecture Notes in Computer Science: Advances in Computing and Information*, vol. 497, pp. 383-394, 1991.

[5] M. Farber and J.M. Keil, Domination in permutation graphs, *Journal of Algorithms*, vol. 6, pp. 309-321, 1985.

[6] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., New York, 1980.

[7] D. Helmbold and E.W. Mayr, Applications of parallel algorithms to families of perfect graphs, *Computing*, vol. 7, pp. 93-107, 1990.

[8] W.L. Hsu, Maximum weight clique algorithms for circular-arc graphs and circle graphs, *SIAM Journal on Computing*, vol. 14, pp. 224-231, 1985.

[9] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[10] D. Kozen, U.V. Vazirani and V.V. Vazirani, NC algorithms for comparability graphs, interval graphs, and testing for unique perfect matching, *Proc. of the 5th Conference on Foundation of Software Technology and Theoretical Computer Science*, New Delhi, pp. 498-503, 1985.

[11] L. Kucera, Parallel computation and conflicts in memory access, *Inform. Process. Lett.*, vol. 14, pp. 93-96, 1982.

[12] J.Y.-T. Leung, Fast algorithms for generating all maximal independent sets of interval, circular-arc and chordal graphs, *Journal of Algorithms*, vol. 5, pp. 22-35, 1984.

[13] S.D. Nikolopoulos, "Constant-time parallel recognition of split graphs", *Inform. Process. Lett.*, vol. 54, pp. 1-8, 1995.

[14] S.D. Nikolopoulos and S.D. Danielopoulos, "Parallel computation of perfect elimination schemes using partition techniques on triangulated graphs", *Computers and Mathematics with Applications*, vol. 29, pp. 47-57, 1995.

[15] S.D. Nikolopoulos, "Cographs and treshold graphs can be recognized in $O(1)$ time by a CRCW-PRAM", *TR-96-6, Department of Computer Science, University of Cyprus*, 1996.

[16] A. Pnueli, A. Lempel and S. Even, Transitive orientation of graphs and identification of permutation graphs, *Canadian J. Math.*, vol. 23, pp. 160-175, 1971.

[17] J. Reif (editor), *Synthesis of Parallel Algorithms*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1993.

[18] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.

[19] J. Spinrad, On comparability and permutation graphs, *SIAM Journal on Computing*, vol. 14, pp. 658-670, 1985.

[20] J. Spinrad, A. Brandstadt and L. Stewart, Bipartite permutation graphs, *Discrete Applied Mathematics*, vol. 18, pp. 279-292, 1987.

[21] K.J. Supowit, Decomposing a set of points into chains, with applications to permutation and circle graphs, *Inform. Process. Lett.*, vol. 21, pp. 249-252, 1985.

[22] K.H. Tsai and W.L. Hsu, Fast algorithms for the dominating set problem on permutation graphs, *Lecture Notes in Computer Science: Algorithms*, vol. 450, pp. 109-117, 1990.

[23] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM Journal on Computing*, vol. 6, pp. 505-517, 1977.

[24] U. Vishkin, Implementation of simultaneous memory address access in model that forbid it, *Journal of Algorithms*, vol. 4, pp. 45-50, 1983.

[25] C-W. Yu and G-H. Chen, Parallel algorithms for permutation graphs, *BIT*, vol. 33, pp. 413-419, 1993.

[26] C-W. Yu and G-H. Chen, Generate all maximal independent sets in permutation graphs, *Intern. J. Computer Math.*, vol. 47, pp. 1-8, 1993.