Mixed Domination in Trees: A Parallel Algorithm

Gur Saran Adhar
Mathematical Sciences Department
Univ. of North Carolina Wilmington
NC 28403, U.S.A.
adhar@seq.cms.uncwil.edu

Shietung Peng
Distributed Parallel Processing Lab.
University of Aizu
Aizu-wakamatsu, Japan
s-peng@rsc.u-aizu.ac.jp

ABSTRACT

A set of vertices S in a graph $G = (V, E)$ is called a *dominating set* of G if every vertex in the set $(V \setminus S)$ is adjacent to some vertex in S. For arbitrary graphs, the problem of computing smallest dominating set is NP-complete [3]. A slightly more general version of this problem is called "mixed domination" problem [1].
In this paper we present new parallel NC algorithm to find smallest mixed dominating set in trees. The algorithm is based on tree compression techniques that have been traditionally used to evaluate linear arithmetic expressions in parallel [2], [4], [5]. In this respect the paper generalizes the application of tree compression techniques to solve combinatorial problems in trees. The model of parallel computation used is the CRCW P-RAM (Concurrent Read Concurrent Write Parallel RAM), where more than one processor can concurrently read from or write into the same memory location during the same memory cycle. Writing conflicts are resolved in a non-deterministic fashion. The algorithm requires $O(n)$ processors and runs in $O(\log n)$ time on a CRCW P-RAM.
Keywords: NC algorithm, domination, tree, graph

1 Introduction:

A set of vertices S in a graph $G = (V, E)$ is called a *dominating set* of G if every vertex in V which is not in S is adjacent to some vertex in S. For arbitrary graphs, the problem of computing smallest dominating set is NP-complete [3]. A slightly more general version of this problem called "mixed domination" problem [1] is stated as follows:

> *Given that the vertices of a graph G are partitioned into three subsets, V_1, V_2, and V_3, find a smallest set S of vertices that contains all of the vertices in V_3 and which dominates all the vertices in V_2. Vertices in V_1 are not required to be dominated, however may be included in S to dominate vertices in V_2.*

The set V_1 is called the set of *free vertices*, V_2 the set of *bound vertices* and V_3 is called the set of *required vertices*.
Solution to mixed domination problem in trees, on a sequential computation model, is known [1]. The linear time sequential algorithm is based on the following theorem:

Theorem 1 [1] Let T be a tree having free, bound and required vertices, V_1, V_2, and V_3 respectively. Let v be an end vertex of T which is adjacent to vertex u. Then mixed domination set $md(T)$ of tree T is given by:

(i). if $v \in V_1$, then $md(T) = md(T - v)$;
(ii). if $v \in V_2$ and T' is the tree which results from deleting v and relabeling u as "required" then $md(T) = md(T')$;
(iii). if $v \in V_3$ and $u \in V_3$ then $md(T) = 1 + md(T - v)$;
(iv). if $v \in V_3$ and $u \notin V_3$ and if T' is the tree which results from deleting v and relabeling u as "free", then $md(T) = 1 + md(T')$; ♣

Sequential algorithm in [1] selects one leaf of the tree and successively applies conditions (i).-(iv). of Theorem 1 in each iteration until the entire tree is processed. This incremental nature of the algorithm is not attractive for parallel implementation.
In this paper we propose a parallel algorithm to solve the mixed domination problem in trees. Our algorithm uses parallel tree compression techniques that have been traditionally used for evaluating arithmetic expressions in parallel [2], [4], [5]. In this sense this paper extends the scope of tree compression techniques to solve combinatorial problems.
The model for parallel computation is CRCW P-RAM; where more than one processor can read from or write into same memory location during one memory cycle. In case of writing conflicts, however, only one processor succeeds and it is not known in advance which one eventually succeeds.

2 Algorithm and Complexity

We formulate the mixed domination problem as a special type of graph *coloring* problem. In this formulation all vertices in a partition are given same color whereas vertices in different partitions receive different colors. For example, let the set of free vertices V_1 be colored green, let the set of bound vertices V_2 be colored blue, and let the set of required vertices V_3 be colored red. then the mixed domination problem can be stated as:
"Given a graph which, initially, has some red vertices, some blue vertices, and some green vertices. Re-color the green and blue vertices in the graph such that every blue vertex is adjacent to some red vertex in the final coloring."
Any coloring of the graph satisfying above condition will be called a *valid* coloring. Among all possible valid colorings the valid coloring which has fewest red vertices will be called *optimum coloring*. Clearly a valid coloring, as defined here, corresponds to a mixed domination set and the optimum coloring corresponds to minimum mixed domination set.
NOTATION: The input tree is considered to be a rooted tree and $(u \rightarrow v)$ is used to denote an edge from u to its parent v in the tree. A *chain* is a set of vertices which have in-degree equal to one. $\mathcal{C}(u)$ is used in following sections to denote the color of a vertex u in the initial coloring and $\mathcal{C}^*(u)$ is used to refer to the color of a vertex u in the optimum coloring. $P(X)$ is used to denote *color* of the parent v of a vertex u which received color X in the initial coloring (i.e. $(u \rightarrow v)$, $\mathcal{C}(u) = X$,

$\mathcal{C}(v) = P(X)$); and $P^*(X)$ is used to denote optimum *color* of the parent v of a vertex u which is colored X, as if X colored vertex was the only child of v (i.e. $(u \rightarrow v)$, $\mathcal{C}^*(u) = X$, $\mathcal{C}^*(v) = P^*(X)$). It should be noted that the $P^*(X)$ notation encompasses the influence of only one child on its parent in the optimum coloring and does not define the final color of $P(x)$ (since it is influenced by all the children).
For example, $P(R)$ denotes color of the parent of a red vertex in the initial coloring, and $P^*(G)$ denotes color of the parent of a green vertex in optimum coloring as if the green vertex was the only child.
It is observed that: (a). since a required vertex in the initial partition must appear in the final dominating set, by definition, a red vertex will never change color; and (b). since a free vertex can either remain free or join the dominating set, by definition, a green vertex can either remain green or become red but never become blue during the re-coloring process. Based on these two observations, we state our first lemma.

Lemma 1 Following statements are true:

$$P(R) = R \quad implies \quad P^*(R) = P(R) = R \tag{1}$$
$$P(R) = G \quad implies \quad P^*(R) = P(R) = G \tag{2}$$
$$P(R) = B \quad implies \quad P^*(R) = G \tag{3}$$
$$P(B) = R \quad implies \quad P^*(B) = P(B) = R \tag{4}$$
$$P(G) = R \quad implies \quad P^*(G) = P(G) = R \tag{5}$$
$$P(G) = G \quad implies \quad P^*(G) = P(G) = G) \tag{6}$$

Proof:
(1). $(P(R) = R)$ *A red vertex with red parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = R$ and $\mathcal{C}(v) = R$. The influence of u on v is known at the outset, since u will never change color. It follows that after removing the edge $(u \rightarrow v)$ the tree T_1 rooted at u and the remaining tree $(T - T_1)$ can be colored seperately. The seperate optimum colorings of T_1 and $(T - T_1)$ give the optimum coloring of T.
(2). $(P(R) = G)$ *A red vertex with green parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = R$ and $\mathcal{C}(v) = G$. The influence of u on v is known at the outset, since u will never change color. It follows that after removing the edge $(u \rightarrow v)$ the tree T_1 rooted at u and the remaining tree $(T - T_1)$ can be separately colored. The seperate optimum colorings of T_1 and $(T - T_1)$ give the optimum coloring of T.
(3). $(P(R) = B)$ *A red vertex with blue parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = R$ and $\mathcal{C}(v) = B$. The influence of u on v is known at the outset, since u will never change color. It follows that after removing the edge $(u \rightarrow v)$ and re-coloring v as green the tree T_1 rooted at u and the remaining tree $(T - T_1)$ can be separately colored. This re-coloring will take into account the influence of a red vertex on its parent. The seperate optimum colorings of T_1 and $(T - T_1)$ give the optimum coloring of T.
(4). $(P(B) = R)$ *A blue vertex with red parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = B$ and $\mathcal{C}(v) = R$. No matter what is the final color of u (whether green, blue, or red) color of v will not change. It follows that after turning u to green and removing the edge $(u \rightarrow v)$ the tree T_1 rooted at u

and the remaining tree $(T - T_1)$ can be separately colored. It is possible to set u free because u is dominated from above by its parent v. The seperate optimum colorings of T_1 and $(T - T_1)$ give the optimum coloring of T.

(5). $(P(G) = R)$ *A green vertex with red parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = G$ and $\mathcal{C}(v) = R$. No matter what is the final color of u (whether green, or red) color of v will not change. It follows that after removing the edge $(u \rightarrow v)$ the tree T_1 rooted at u and the remaining tree $(T - T_1)$ can be separately colored. The seperate optimum colorings of T_1 and $(T - T_1)$ give the optimum coloring of T.

(6). $(P(G) = G)$ *A green vertex with green parent*:
Let $(u \rightarrow v)$ be an edge such that $\mathcal{C}(u) = G$ and $\mathcal{C}(v) = G$. A green vertex can either retain its original color or become red. A green vertex never becomes blue. No matter what is the final color of u, (whether green, or red), the color of v will not be influenced by the final color of u. It follows that after removing the edge $(u \rightarrow v)$ the tree T_1 rooted at u and the remaining tree $(T - T_1)$ can be colored separately. The seperate optimum colorings of T_1 and $(T - T_1)$ also give the optimum coloring of T. ♣

The proof of the Lemma 1 forms the basis of the following tree_cuttting algorithm. The edges identified in the proof of the lemma cut the tree into two or more smaller trees. Each smaller tree will be processed seperately and in parallel.

Tree_Cutting (T)

1. For every directed edge $(u \rightarrow v)$ **do in parallel**
   Case {
   $(\mathcal{C}(u) = R) \wedge (\mathcal{C}(v) = R)$: remove $(u \rightarrow v)$;
   $(\mathcal{C}(u) = R) \wedge (\mathcal{C}(v) = G)$: remove $(u \rightarrow v)$;
   $(\mathcal{C}(u) = G) \wedge (\mathcal{C}(v) = G)$: remove $(u \rightarrow v)$;
   $(\mathcal{C}(u) = G) \wedge (\mathcal{C}(v) = R)$: remove $(u \rightarrow v)$;
   $(\mathcal{C}(u) = R) \wedge (\mathcal{C}(v) = B)$: remove $(u \rightarrow v)$; $\mathcal{C}(v) = G$; /*re-color v green */
   $(\mathcal{C}(u) = B) \wedge (\mathcal{C}(v) = R)$: remove $(u \rightarrow v)$; $\mathcal{C}(u) = G$; /*re-color u green */
   }
2. Find connected trees in T after edges in step-1 are removed.

end.

At the end of the Tree_Cutting(T) all the edges $(u \rightarrow v)$ within any connected tree, have one of the following colorings: $(\mathcal{C}(u) = B, \mathcal{C}(v) = G)$, $(\mathcal{C}(u) = B, \mathcal{C}(v) = B)$, or $(\mathcal{C}(u) = G, \mathcal{C}(v) = B)$. The leaf vertices are blue, and the internal vertices are either green or blue. Each of the connected tree is now processed separately and in parallel. The main algorithm mixed_domination proceeds to process a connected tree in a manner similar to tree contraction algorithm of Miller and Reif [2]. In the tree contraction algorithm of [2] two operations namely: tree-rake; and tree-compress; are applied repeatedly until the entire tree is processed. Tree-rake operation removes leaves whereas the tree-compress operation compresses long chains of vertices in a tree.

We modify the tree-rake function to invoke Tree_Cutting algorithm in addition to the standard rake operation. The modified tree-rake operation is implemented in the following function.

Tree_rake(T)

1. For every leaf u, with parent v **do in parallel**
   Case {
   $(\mathcal{C}(u) = R) \wedge (\mathcal{C}(v) = R)$: remove u;
   $(\mathcal{C}(u) = R) \wedge (\mathcal{C}(v) \neq R)$: assign color G to v; remove u;
   $(\mathcal{C}(u) = G) : remove$u;
   (C(u) = B): assign color R to v and remove u;
   }
2. Tree_Cutting(T);

end.

Tree compress operation is described now in the context of the mixed domination problem. Following lemma states conditions for optimum coloring of a chain of vertices.

Lemma 2 *There is an optimum coloring of a chain, having blue and green vertices, which satisfies following conditions:*

(i). within a chain of two or more consecutive green vertices, at most one green vertex can change color to red.
(ii). within a chain of three consecutive blue vertices, exactly one vertex must receive red color.
(iii). within a chain of two consecutive blue vertices followed by a green vertex, exactly one vertex must receive red color.

♣

Condition-(i) in Lemma 2 suggests that: a chain of green vertices can be compressed into a single green vertex; and the resulting compressed path can be colored without affecting optimum coloring of the original path. This is possible since at most one green vertex within a chain of green vertices can receive red color. Whenever a chain of green vertices is compressed into a single vertex a special ($\bigoplus$) marker is inserted to record the fact.
Condition-(ii) in Lemma 2 suggests that chains with color patterns of type "blue-blue-blue" can be eliminated. In the case when a "blue-blue-blue" chain is eliminated by the application of condition-(ii) special ($) -marker is inserted in the path to record this fact so that a red vertex is include in the final optimum coloring. Similarly condition-(iii) in Lemma 2 suggests that chains with color patterns of type "blue-blue-green" can be eliminated as well. When a "blue-blue-green" chain is eliminated by the application of condition-(iii) special (#)-marker is inserted in the path to record this fact so that a red vertex is include in the final optimum coloring. Shortcut operations suggested by condition-(i), (ii), and (iii) of Lemma 2 on a chain are illustrated in Figure 1.
The shortened chain after compression will have alternate green and blue vertices only since all sequences with two or more blue vertices have been eliminated and replaced by markers similarly all sequences with two or more green vertices have been replaced with one green vertex. In addition to vertices ($\bigoplus$), (#), ($)-markers appear embedded in between. Following lemmas state conditions for optimum coloring of a chain, delimited by ($\bigoplus$)-markers, having alternate blue and green vertices.

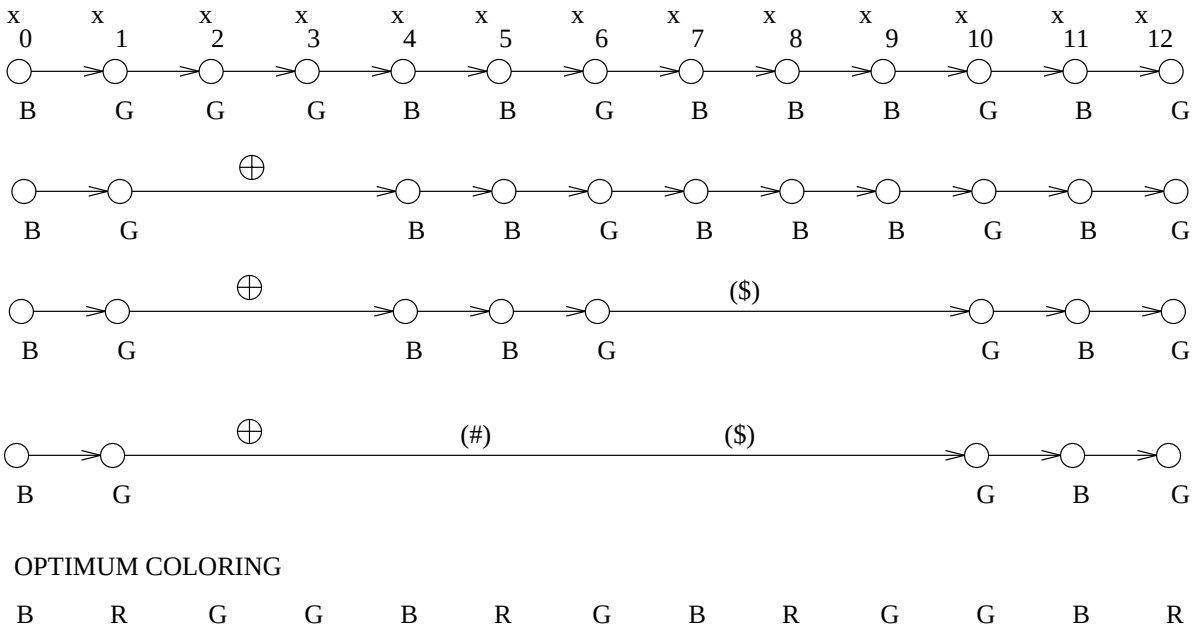

Figure 1: shortcut operation

Lemma 3 *There is an optimum coloring of a chain which has alternate blue and green vertices that satisfies following condition:*
Number of red color vertices r^* in optimum coloring is:

$$r^* \leq \begin{cases} (m+1) & \text{if } n = (2m+1) \\ m & \text{if } n = 2m \end{cases}$$

where n is the number of blue vertices. ♣

While coloring a chain having alternate blue and green vertices a special case arises which is characterized by following lemma.

Lemma 4 *If a green vertex u receives red color during optimum coloring of a chain which has alternate blue and green vertices, then vertex v is colored green if a (#)-marker or a ($)-marker appears on $(u \rightarrow v)$.* ♣

The operation suggested by Lemma 4 is illustrated in Figure 2. We say that, the influence of red vertex propagates thru (#)-marker and ($)-marker. In the example chain the influence of x_1 propagates thru $x_2, x_3, x_4, x_5, x_6, x_7$ and turns x_8 green.

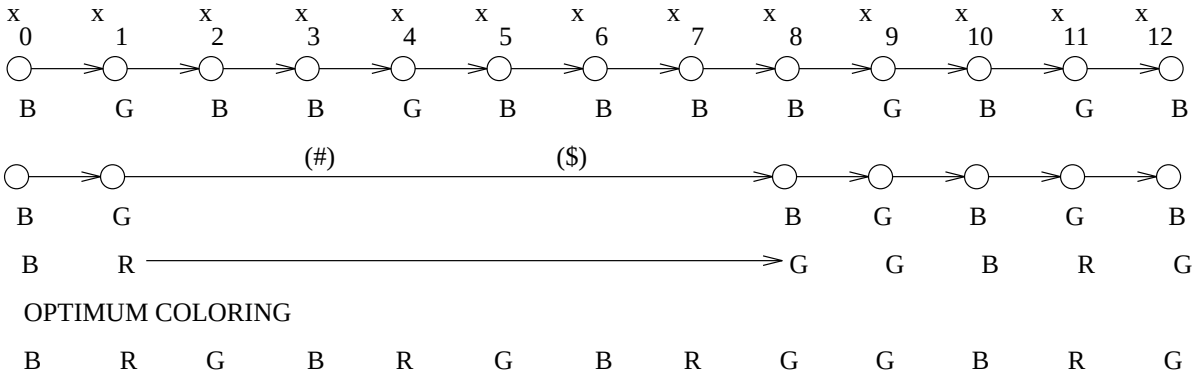

Figure 2:

Algorithm Tree_Compress()
begin

1. For every maximal chain **do in parallel**
   {
   1. Replace chain of green vertices $(G - G - \ldots - G)$ by one (G) vertex and place a $(\bigoplus)$-marker next to it (Lemma 2.(i)).
   2. Shortcut chain of vertices which have color pattern $(B - B - B)$, and insert a ($)-marker in its place (Lemma 2.(ii)).
   3. Shortcut chain of vertices which have color pattern $(B - B - G)$, and insert a (#)-marker in its place (Lemma 2.(iii)).
   4. Color the chain $(B - G - B - G - B - G \ldots B - G)$ by fewest red color, (Lemma 3, and Lemma 4).

   }

end.

COMPLEXITY: Step 1, 2 and step 3 can be completed with $O(n)$ processors in $O(\log n)$ time. The task in each step is to find all occurrences of a key pattern in a string. Step 4 also has the same processor and time complexity to propagate by doubling. Step 5 can be accomplished in constant time with $O(n)$ processors. The overall complexity is therefore $O(n)$ processors and $O(\log n)$ time.

3 Conclusion

We have presented a parallel NC algorithm to solve mixed domination problem in trees which requires $O(n)$ processors and runs in $O(\log n)$ time on a CRCW P-RAM model. Sequential algorithm for the problem runs in $O(n)$ time. The total effort by the parallel algorithm is logarithmic multiple of the work by sequential algorithm. In this sense our algorithm is log-optimal.

References

[1] W. Cockayane, S. Goodman, and S. Hedetniemi, A Linear Algorithm for the Domination Number of a tree, Information Processing Letters, Vol. 4 1975, pp 41-44.

[2] G. E. Miller, and J.H. Reif, Parallel Tree Contraction and its Applications, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp 478-489.

[3] M. R. Garey and D. S. Johnson, Computers and Interactability: A guide to the theory of NP-Completeness, (Freeman, San Francisco, CA 1979).

[4] Richard Cole and Uzi Vishkin, The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time, Ultracomputer Note -108, TR-242, Department of Computer Science, Courant Institute NYU, 1986.

[5] K. Abrahamson, N. Dadoun, A simple tree contraction algorithm, Department of Computer Science, The University of British Columbia, Tech. Report 87-30, August 87.