In SIAM JOURNAL ON COMPUTING, 1996. Copyright SIAM

AN EFFICIENT PARALLEL ALGORITHM FOR THE GENERAL PLANAR MONOTONE CIRCUIT VALUE PROBLEM*

VIJAYA RAMACHANDRAN† AND HONGHUA YANG†

Abstract. A planar monotone circuit (PMC) is a Boolean circuit that can be embedded in the plane and that contains only AND and OR gates. Goldschlager, Cook & Dymond and others have developed NC^2 algorithms to evaluate a special layered form of a PMC. These algorithms require a large number of processors ($\Omega(n^6)$, where n is the size of the input circuit). Yang, and more recently, Delcher & Kosaraju have given NC algorithms for the general planar monotone circuit value problem. These algorithms use at least as many processors as the algorithms for the layered case.

This paper gives an efficient parallel algorithm that evaluates a general PMC of size n in polylog time using only a linear number of processors on an EREW PRAM. This parallel algorithm is the best possible to within a polylog factor, and is a substantial improvement over the earlier algorithms for the problem. The algorithm uses several novel techniques to perform the evaluation, including the use of the dual of the plane embedding of the circuit to determine the propagation of values within the circuit.

Key words. circuit value problem, planar monotone circuit, plane graph, dual graph, parallel algorithm, EREW PRAM

AMS subject classifications. 68Q10, 68Q15, 68Q20, 68Q22, 68Q25, 68R10, 05C10

1. Introduction.

A *Boolean circuit* is a directed network of AND, OR and NOT gates whose wires do not form directed cycles. The problem of evaluating a Boolean circuit, given the values of its inputs, is called the *circuit value problem* (CVP). This is a central problem in the area of algorithms and complexity. Ladner [16] has shown that CVP is P-complete under log space reductions. Some special cases of CVP have been studied, among which the *monotone circuit value problem*, where the Boolean circuit has only AND and OR gates, and the *planar circuit value problem*, where the Boolean circuit has a plane embedding, have been shown to be P-complete by Goldschlager [10].

A *planar monotone circuit* (PMC) is a Boolean circuit that is both planar and monotone. One interesting special case of CVP is the *planar monotone circuit value problem* (PMCVP), which is the problem of evaluating a PMC. In this paper we give an efficient parallel algorithm for the PMCVP that runs in polylog time using a linear number of processors. The parallel computation model we use here is the EREW PRAM model [14].

Here is a summary of earlier results for the PMCVP. Goldschlager [7, 8], Dymond & Cook [4], and Mayr [17] have shown that the problem of evaluating a special layered form of PMC is in NC^2. The first NC algorithm for the general PMCVP was given in Yang [24]; this algorithm runs in $O(\log^3 n)$ time on an EREW PRAM, and uses the straight-line code parallel evaluation technique of Miller, Ramachandran & Kaltofen [18]. Recently Delcher & Kosaraju [3] have given another NC algorithm for the general PMCVP that runs in $O(\log^4 n)$ time using a polynomial number of processors on a CREW PRAM. All of the algorithms listed above use a large number of processors (at least $\Omega(n^6)$, where n is the size of the input circuit).

* This work was supported in part by Texas Advanced Research Projects Grant 003658480 and NSF Grant CCR 90-23059. An extended abstract of this work appears in [21].

† Department of Computer Sciences, University of Texas at Austin, Austin, Texas, 78712-1188 (vlr@cs.utexas.edu, yanghh@cs.utexas.edu).

In earlier work (Ramachandran & Yang [20]) we gave an $O(\log^2 n)$ time EREW PRAM algorithm using a linear number of processors to evaluate a layered PMC. The algorithm we present in this paper, when restricted to evaluate a layered PMC, works with the same processor-time bounds as the one in [20]; however, it is substantially different in that it works on a plane embedding of the PMC and its dual graph instead of exploiting a nice layered structure as in [20]. In one sense our algorithm can be considered to be simpler than the one in [20] since our new approach allows us to eliminate some tedious case analysis used in [20]. Our algorithm uses some ideas from [20], as well as from [24] and [3]. In the highest level of our algorithm, we use an approach similar to that used in [3] to transform a general PMC into 'face f induced subcircuits' (using the terminology of [24], – these circuits are called 'focused circuits' in [3]). These subcircuits are then evaluated using an algorithm to evaluate a 'one-input-face PMC'. The major contribution of our paper is our efficient parallel algorithm to evaluate a one-input-face PMC, which is a PMC, not necessarily layered, all of whose input nodes are on the boundary of one face.

The rest of this paper is organized as follows. In Section 3, we present our algorithm to evaluate a one-input-face PMC. The treatment in Section 3 is self-contained and does not depend on any result in [20]. In Section 4 we give an algorithm that runs in polylog time using n processors on an EREW PRAM for evaluating a face f induced subcircuit given a special type of an input assignment. This algorithm works by recursively applying the algorithm for evaluating a one-input-face PMC. Finally, in Section 5 we give an algorithm that runs in polylog time using n processors on an EREW PRAM for solving the general PMCVP by recursively applying the algorithm for evaluating a face induced subcircuit.

Our results are of interest for several reasons. In designing our efficient parallel algorithm for the PMCVP we have developed a variety of efficient parallel algorithms for processing planar DAGs, especially the technique of working on the dual of a planar DAG. (Other examples of algorithmic techniques based on the dual of a plane embedding can be found in [22, 13].) These tools are likely to be of use in algorithms for other problems on planar directed graphs. Our results are of interest in the context of parallel complexity since all of the earlier algorithms for the PMCVP used indirect methods such as the relationship between sequential space and parallel time [1] or the parallel evaluation of straight-line code [18] to place the problem in NC. By using direct techniques, not only are we able to place the problem in NC, but we are able to obtain a very efficient algorithm for its solution. Finally, the evaluation of circuits is a basic and important problem in computer science. Planar circuits occur very naturally in the design of integrated circuits, and the requirement that the circuit be monotone is not a restriction if the inputs are available together with their complements. Thus our efficient parallel algorithm for the evaluation of planar monotone circuits could be of practical importance.

2. Preliminaries.

DEFINITION 2.1. A *face* of a plane graph $C = (V, E)$ is a maximal portion of the plane for which any two points may be joined by a curve such that each point of the curve neither corresponds to a vertex of C nor lies on any curve corresponding to an edge of C. The *boundary* of a face f in C consists of all those points x corresponding to vertices and edges of C having the property that x can be joined to a point of f by a curve, all of whose points different from x belong to f. (By this definition, a single edge in f belongs to the boundary of f.)

DEFINITION 2.2. An embedded *planar monotone circuit (PMC)* is a plane directed acyclic graph (DAG) $C = (V, E)$, where

(i) V is the set of gates (or vertices) in the PMC consisting of input nodes, AND gates, and OR gates,

(ii) E is the set of directed wires (or edges) in the PMC,

(iii) the fan-in (or in-degree) of an input node is 0, of an AND or OR gate is either 1 or 2, C may have input nodes that are in different faces,

(iv) the fan-out (or out-degree) of an output gate is 0, of other gates is nonzero, C may have more than one output gate, but all output gates of C are in the same face.

In the rest of the paper, whenever we use the term PMC, we should assume that the PMC is given with an embedding. In case an embedding is not given, we can use the algorithm in Ramachandran & Reif [19] to obtain one. We assume that the plane embedding of a PMC C is given by its combinatorial definition: a clockwise cyclic ordering of edges incident to each vertex in C, and a counterclockwise cyclic ordered sequence of vertices and edges $g_0, e_0, g_1, e_1, \ldots, g_{k-1}, e_{k-1}$ on the boundary of a face in C such that for any i, $0 \leq i \leq k-1$, the edges e_{i-1} and e_i are incident to vertex g_i and e_{i-1} appears immediately before e_i in the cyclic ordering of the edges incident to g_i.

DEFINITION 2.3. A *complete input assignment* to a PMC is an assignment of values 0 or 1 to all input nodes in the PMC. A *partial input assignment* to a PMC is an assignment of values 0 or 1 to a subset of the input nodes in the PMC. An input node that is not assigned a value in a partial input assignment has an *unknown* value. (A complete input assignment is a special case of a partial input assignment.)

DEFINITION 2.4. The *partial evaluation problem of a PMC* is the problem of evaluating the value of every gate in the PMC that can be evaluated, given a partial input assignment to the PMC. The gates in a PMC that cannot be evaluated under a partial input assignment have *unknown* values. A PMC is *completely evaluated* if the value of every gate in it is either 0 or 1, it is *partially evaluated* otherwise. The *planar monotone circuit value problem (PMCVP)* is the problem of completely evaluating a PMC, given a complete input assignment to the PMC.

A one-input-face PMC we define below is a PMC with the following differences: i) It is a restriction of a PMC in that all of its input nodes are on the boundary of a single face. ii) It is a generalization of a PMC in that it may contain *pseudo wires*, which are wires that carry no value. Our algorithm may need to add pseudo wires in a PMC during the computation in subsection 3.2.

DEFINITION 2.5. A *one-input-face PMC C* is a variant of a PMC with the following properties.

1. C is a plane DAG consisting of input nodes, AND gates, and OR gates.

2. All input nodes of C are on the boundary of a single face f_I. The in-degree of an input node is 0, of an AND or OR gate is either 1 or 2.

3. A gate in C with out-degree 0 is called an *output gate*. The out-degree of other gates or input nodes is nonzero. C may have more than one output gate, but all output gates of C are on the boundary of a single face f_O.

4. If f_I and f_O are identical, then the input nodes and the output gates of C may not interlace, i.e. there exists a part of the boundary of f_I which contains all input nodes but no output gates.

5. Some of the gates in C may contain a single output wire that does not carry any value and that goes into a two-input gate (note that such a gate with a single

output wire that does not carry any value is *not* an output gate). We call a wire that does not carry any value a *pseudo wire*. Further, a two-input gate g may receive at most one input from a pseudo wire, and the value of g only depends on its non-pseudo input wire(s).

We will give a recursive algorithm in Section 3 that evaluates a one-input-face PMC of size n in $O(\log^2 n)$ time using n processors on an EREW PRAM, where properties 4 and 5 in Definition 2.5 are needed after the first level of recursion.

DEFINITION 2.6. $Reach(i_1, \ldots, i_k)$ for some input nodes $i_1, \ldots, i_k$ in a PMC is the part of the PMC that is reachable from $i_1, \ldots, i_k$. Given a subcircuit P of a PMC, $Reach(P)$ is defined to be the part of the circuit reachable from the input nodes in P. $Induced(i_1, \ldots, i_k)$ for some input nodes $i_1, \ldots, i_k$ that are on the boundary of a single face f (where $i_1, \ldots, i_k$ need not be all the input nodes of f) in a PMC C is $Reach(i_1, \ldots, i_k)$ augmented with a new input node set V_{IN} and a wire set E_{IN} which are formed as follows: if a gate $x \in C \setminus Reach(i_1, \ldots, i_k)$ has some output wires $(x, y_1), (x, y_2), \ldots, (x, y_l)$, $(l \geq 1)$ pointing to gates $y_1, y_2, \ldots, y_l$ in $Reach(i_1, \ldots, i_k)$, then we add a new input node i_x to V_{IN} and wires $(i_x, y_1), (i_x, y_2), \ldots, (i_x, y_l)$ to E_{IN}. We call such $Induced(i_1, \ldots, i_k)$ a face f *induced (sub)circuit*.

It is easy to see that a face f induced circuit is still a PMC. A face f induced circuit C_f is not necessarily a one-input-face PMC since the newly added input nodes can appear in faces other than f in C_f. But C_f is still simpler than a general PMC in the sense that all gates in C_f except the newly added input nodes are reachable from some input nodes on the boundary of face f, and once the values of the new input nodes are known, C_f can be transformed into a logically equivalent one-input-face PMC. We give an algorithm in Section 4 that partially evaluates a face induced circuit of size n given a special input assignment, in $polylog(n)$ time using n processors on an EREW PRAM, by recursively calling the algorithm for evaluating a one-input-face PMC. In Section 5, we give an algorithm that completely evaluates a general PMC of size n in $polylog(n)$ time using n processors on an EREW PRAM, by recursively calling the algorithm for partially evaluating a face induced circuit, given a special input assignment.

3. The One-Input-Face PMC. We first consider the problem of completely evaluating a one-input-face PMC C given a complete input assignment. This treatment appears in subsections 3.1, 3.2, and 3.3. We then solve the problem of partially evaluating a one-input-face PMC in subsection 3.4. Our approach is to first find a set of gates in C that are guaranteed to have value 1, and then recursively evaluate the remaining smaller unevaluated subcircuits of C. In an earlier paper [20] we had considered a special case of a one-input-face PMC, namely, a layered PMC (as mentioned in the introduction). In a layered PMC, a sequence of gates with value 1 at one layer guarantees that a sequence of gates at the next layer will have value 1; and the left and the right boundaries of the gates with value 1 are defined by the starting gate and the ending gate of the sequence at each layer respectively. In a one-input-face PMC C that does not have the layered property, we do not have such a simple correspondence. In the treatment below, we work with the dual of a plane embedding of C, and define the left and right boundaries of the gates with value 1 in a manner that allows us to determine the propagation of the 1 values through the circuit.

In subsection 3.1, we will give some definitions and lemmas. In subsection 3.2, we will present our techniques for finding the left and right boundaries of the gates with value 1 and for simplifying the remaining circuit of C. In subsection 3.3, we will

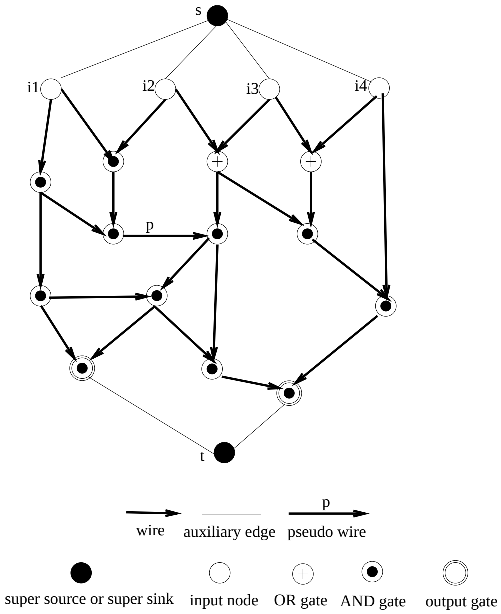

FIG. 1. *C_{aug}: a one-input-face PMC C augmented with a super source s and a super sink t. Here $f_I = f_O$.*

give the complete algorithm for evaluating a one-input-face PMC given a complete input assignment and its complexity analysis. In subsection 3.4, we will extend the algorithm to evaluate a one-input-face PMC given a partial input assignment.

Throughout Section 3, we use C to refer to a one-input-face PMC unless otherwise stated.

3.1. Definitions.

DEFINITION 3.1. A gate is a *source* of a face f in C if it has two output wires that are on the boundary of f. A gate is a *sink* of a face f in C if it has exactly two input wires and both are on the boundary of f. Let f_I be the face of C whose boundary contains all the input nodes and let f_O be the face of C whose boundary contains all the output gates. C_{aug} is a DAG obtained from C by adding a super source s in f_I and auxiliary wires connecting s to every input node in C, and a super sink t in f_O and auxiliary wires connecting every output gate to t (see Figure 1).

By Definition 2.5, if f_I and f_O are identical then the input nodes and the output gates may not interlace in C. Hence C_{aug} is still a plane graph. In the rest of the paper, we assume that C has at least two input nodes. If C has only one input node, then by Definition 2.2 all gates in C have the same value as the only input node, which is a trivial case. The reason that we augment C to C_{aug} is that there is a single source and a single sink in every face of C_{aug} as shown in the following lemma. This property is crucial to many definitions given in this subsection.

Note that not every one-input-face PMC has a downward plane drawing [11] as in Figure 1, if f_I and f_O are different faces.

LEMMA 3.1. *Every face in C_{aug} has exactly one source and one sink.*

Proof. Every gate in C_{aug} is reachable from s and reachable to t. Since C_{aug} is a DAG, there is at least one source and one sink on the boundary of a face in C_{aug}.

Suppose a face f has two sources s_1 and s_2 and therefore two sinks t_1 and t_2. Then we consider the following four directed paths in C_{aug}: the path P_1 from s to s_1, the path P_2 from s to s_2, the path P_3 from t_1 to t, and the path P_4 from t_2 to t. The path P consisting of P_1 and P_2 joins the face f' at s_1 and s_2. The path Q consisting of P_3 and P_4 joins the face f' at t_1 and t_2. But s_1 and s_2 interlace with t_1 and t_2 on the boundary of f. But the two paths P and Q have to be embedded in one side of the boundary of f. A contradiction. Hence every face in C_{aug} has exactly one source and exactly one sink. □

The following lemma is needed for Definition 3.3.

LEMMA 3.2. *The output wires of a gate g in C_{aug} are placed consecutively in the cyclic ordering of the wires around g.*

Proof. Let g be a gate in C_{aug} with two input wires i_1 and i_2 and two output wires o_1 and o_2, such that o_1 and o_2 interlaces with i_1 and i_2 in the cyclic ordering of the wires around g. Since every gate in C_{aug} is reachable to t, there are two directed paths P_1 and P_2 in C_{aug} from g to t, where P_1 goes through o_1 and P_2 goes through o_2. Let x be the first gate (except g) on P_1 that is also on P_2. Since C_{aug} is acyclic, x must be the first gate (except g) on P_2 that is also on P_1. The subpath of P_1 from g to x and the subpath of P_2 from g to x form an undirected cycle which divides the plane into two parts C_{inside} and $C_{outside}$, where C_{inside} is the part of the plane that is inside the cycle and $C_{outside}$ is the part of the plane that is outside the cycle. Without loss of generality, assume that the super source s and the input wire i_2 are in $C_{outside}$. Let $i_1 = (g_1, g)$. Then g_1 and i_1 are in C_{inside} and g_1 is not reachable from s since otherwise there would be a directed cycle in C_{aug}. Hence g_1 is reachable from some input nodes in C_{inside} that cannot be reached from s in $C_{outside}$. A contradiction. □

DEFINITION 3.2. The *left (right) input wire* of a two-input gate g is the input wire of g that appears immediately after (before) an output wire of g in the clockwise cyclic ordering of the wires around g.

Note that the source and the sink of a face f in C_{aug} partition the boundary of f into two disjoint (except at the source and the sink) directed paths.

DEFINITION 3.3. The path from the source to the sink going through the left input wire of the sink is *the counterclockwise boundary* of the face, and the path from the source to the sink going through the right input wire of the sink is *the clockwise boundary* of the face.

DEFINITION 3.4. The *dual digraph* $C^*_{aug} =< V^*, E^* >$ of the plane directed *primal graph* $C_{aug} =< V, E >$ is defined as follows.

(i) For each *primal face* f in C_{aug}, define a *dual vertex* f^* in V^*.

(ii) For each *primal edge* e in C_{aug} such that e is on the clockwise boundary of a primal face f_1 and the counterclockwise boundary of another primal face f_2, define a *counterclockwise dual edge* $e^{*+} = (f_1^*, f_2^*)$ in E^* and a *clockwise dual edge* $e^{*-} = (f_2^*, f_1^*)$ in E^*.

Note that in the dual graph C^*_{aug}, we introduce dual edges of both directions, clockwise and counterclockwise for each edge in the primal graph. The dual graph C^*_{aug} can also be viewed as the result of forming the undirected dual graph of C_{aug} and replacing the dual edges by dual arcs of both directions.

For convenience, for a primal face f, and a primal edge e, we will use f^*, e^{*+}, e^{*-} to indicate the dual vertex of f, the counterclockwise dual edge of e, and the clockwise dual edge of e respectively.

In the following definition, we define an auxiliary graph (which can be viewed as

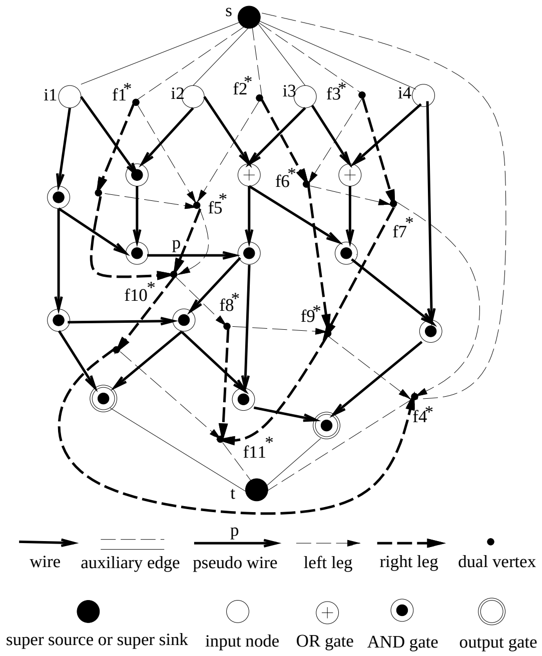

FIG. 2. *C_{aug} and A_{aug}, the auxiliary dual graph. C_{aug} consists of the solid edges. A_{aug} consists of the dashed edges. The two graphs overlap at s and t.*

a subgraph of C^*_{aug} augmented with s and t) that contains some edges called *left legs* and *right legs*. These edges aid us in defining regions of C_{aug} where value 1 propagates from input nodes. Thus, their definitions are dependent on whether a gate is an AND gate or an OR gate and whether a wire is a pseudo wire, since an AND gate has value 1 if both of its inputs have value 1, an OR gate has value 1 if one of its inputs has value 1, and a pseudo wire does not pass any value.

DEFINITION 3.5. The *auxiliary dual graph* $A_{aug} =< V^*_1, E^*_1 >$ is defined as follows (see Figure 2).

(i) V^*_1 contains the dual vertices of all the primal faces in C_{aug}, together with the super source s and the super sink t.

(ii) E^*_1 contains all the dual edges called the *left legs* and the *right legs* defined as follows.

Let f be a primal face in C_{aug} whose boundary does not contain t and let g be the sink of f with left input wire w_l and right input wire w_r.

(i) If g is an OR gate with no pseudo input wire, then the *left leg* and the *right leg* of f^* are w^{*-}_l (i.e., the clockwise dual edge of w_l) and w^{*+}_r (i.e., the counterclockwise dual edge of w_r) respectively.

(ii) If g is an AND gate with no pseudo input wire, then the *left leg* and the *right leg* of f^* are w^{*+}_r and w^{*-}_l respectively.

(iii) If w_l is a pseudo wire, then both the *left leg* and the *right leg* of f^* are w^{*-}_l.

(iv) If w_r is a pseudo wire, then both the *left leg* and the *right leg* of f^* are w^{*+}_r.

(v) Let f be a primal face whose boundary contains t or s in C_{aug}. Then we add an auxiliary edge (f^*, t) or (s, f^*) to E^*_1, respectively.

$A_{aug} \setminus \{s, t\}$ is a subgraph of C^*_{aug} that contains some dual edges of the input wires to the sinks of the faces in C_{aug}. It is easy to see that after being augmented with s and t, A_{aug} is still a plane graph since the input nodes and the output gates of C do not interlace.

LEMMA 3.3. *If there is an edge* $e^* = (f_1^*, f_2^*)$ *in* A_{aug} *which is either a left leg or a right leg, then there is a directed path of length at least 1 from the sink* s_1 *of* f_1 *to the sink* s_2 *of* f_2 *in* C_{aug}.

Proof. By Definition 3.5, s_1 must be on either the clockwise boundary or the counterclockwise boundary of f_2 in C_{aug}. Since a gate (except t) has at most two input wires and therefore can be the sink of at most one face, s_1 cannot be the sink of f_2. Hence there is a directed path of length at least 1 from s_1 to s_2. ∎

COROLLARY 3.3.1. A_{aug} *is a plane DAG whose only vertex with out-degree 0 is* t.

Proof. By Lemma 3.3, A_{aug} is acyclic and hence a DAG. It is obvious that the only vertex in A_{aug} with out-degree 0 is t. ∎

DEFINITION 3.6. Two input nodes i_1 and i_2 in C_{aug} are *adjacent* if the wire (s, i_1) and the wire (s, i_2) are adjacent in the cyclic ordering of the wires around s in C_{aug}. Given a complete input assignment to the input nodes of C_{aug}, a *valid base* B is a maximal sequence of adjacent input nodes with value 1. The *left (right) bounding face* of a valid base B is the face in C_{aug} whose clockwise (counterclockwise) boundary contains an input node in B, but whose counterclockwise (clockwise) boundary does not (see Figure 5).

If all input nodes in C_{aug} have value 1, then the left bounding face and the right bounding face are not defined. But this is a trivial case since we know that all gates of C_{aug} must have value 1.

DEFINITION 3.7. For a valid base B in C_{aug}, let f_l and f_r be the left and right bounding faces of B respectively. The *left boundary* and the *right boundary* of B are the two directed paths P_l^* and P_r^* respectively in A_{aug}, such that (1) P_l^* and P_r^* start from s, (2) P_l^* consists of left legs and auxiliary edges and goes through (s, f_l^*), (3) P_r^* consists of right legs and auxiliary edges and goes through (s, f_r^*), (4) P_l and P_r end at their first common vertex g^* (g^* could be t) after s (see Figures 5 & 6).

DEFINITION 3.8. Given a valid base B, the left boundary P_l^* of B and the right boundary P_r^* of B divide the plane into two regions. The region whose counterclockwise boundary is P_l^* and whose clockwise boundary is P_r^* is called the *internal region* of B, the other region is called the *external region* of B (see Figures 5 & 6).

LEMMA 3.4. *Given a complete input assignment to* C_{aug} *where there is only one valid base* B *(all other input nodes have value 0), a gate* g *in* C_{aug} *evaluates to* 1 *iff* g *is in the internal region of* B.

Proof. Let us embed A_{aug} and C_{aug} in the plane simultaneously with the same super source s and the same super sink t (as in Figure 2) such that the only primal edges in C_{aug} that cross the left and right boundaries P_l^* and P_r^* of B are the input wires to the sinks of some of the primal faces in C_{aug}. (This is provable by Definitions 3.5 and 3.7.) A sink in C_{aug} with a pseudo input wire can have only its pseudo input wire (but not the other input wire) crossing P_l^* or P_r^*. Further, a sink of a primal face f in C_{aug} cannot have its two input wires crossing an edge w_l^* in P_l^* and an edge w_r^* in P_r^* respectively, since otherwise f^* would be the common starting vertex of both w_l^* and w_r^* and therefore would be a common vertex of P_l^* and P_r^*, but P_l^* and P_r^* do not end at f^* since w_l^* is in P_l^* and w_r^* is in P_r^*, a contradiction.

Therefore, if we remove all wires that cross P_l^* and P_r^* from C_{aug} and call the resulting graph C'_{aug}, then every gate (except the input nodes) in C'_{aug} still has at least one non-pseudo input wire (by the previous paragraph), and hence can still be reached from some input nodes without going through pseudo wires. Further, the gates in the internal (external) region of B in C'_{aug} can be reached only by the input nodes in the internal (external) region. Therefore, if we remove all wires crossing P_l^* and P_r^* from C_{aug}, the gates in the internal region of B will have value 1 and the gates in the external region of B will have value 0. We now show that this is still the case even if we do not remove the wires crossing P_l^* and P_r^* from C_{aug}. By Definition 3.5, a wire in C_{aug} outgoing from a gate in the external region of B and incoming to a gate in the internal region of B is either an input wire to a two-input OR gate, or a pseudo input wire to a two-input gate. Hence the gates in the internal region of B in C_{aug} will still have value 1. A primal edge in C_{aug} outgoing from the internal region of B to the external region of B is either an input wire to a two-input AND gate that is in the external region of B, or a pseudo wire to a two-input gate that is in the external region of B. Hence the gates in the external region of B in C_{aug} will still have value 0. Hence the lemma holds. □

COROLLARY 3.4.1. *Given a complete input assignment to* C_{aug}*, if a gate* g *is in the internal region of a valid base in* C_{aug}*, then* g *evaluates to* 1.

Proof. By Lemma 3.4 and the monotonicity of C_{aug}. □

Note that the reverse of Corollary 3.4.1 need not be true if there is more than one valid base in C_{aug}, i.e., some gates outside the internal regions of the valid bases of C_{aug} might also be evaluated to 1. Hence our approach to evaluate a one-input-face PMC C_{aug} is to first find some of the gates that evaluate to 1 based on the internal regions of the valid bases of C_{aug}, remove them from C_{aug}, and repeatedly evaluate the resulting C_{aug}.

3.2. Complete Evaluation of a One-Input-Face PMC. In this subsection, we give an efficient method for computing the left and the right boundaries for all valid bases in C_{aug} simultaneously, given a complete input assignment to C_{aug}. The main idea is to identify for each dual vertex f^* in A_{aug} whether it is on the left (right) boundary of some valid base in C_{aug} (i.e., whether $BOUN_l(f^*)$ or $BOUN_r(f^*)$ defined in Definition 3.10 below is nonempty). Based on this approach, we present a technique to transform the part of C_{aug} that has not been evaluated to 1 into several subcircuits that are one-input-face PMCs with smaller sizes, and more importantly, with geometrically decreasing number of valid bases.

We first define two tree structures that consist of left legs and right legs.

DEFINITION 3.9. Let V_l (V_r) be the set of vertices of A_{aug} (except s) that are reachable through left (right) legs and auxiliary edges incoming to t from the dual vertex of the left (right) bounding face of some valid base in C. We define T_l^* (T_r^*) to be the subgraph of A_{aug} induced by V_l (V_r).

For example, Figure 3 gives T_l^* and T_r^* for the circuit in Figure 1 with an input assignment $(0, 1, 0, 1)$ to the input nodes i_1, i_2, i_3, i_4.

LEMMA 3.5. *Both* T_l^* *and* T_r^* *are convergent trees.*

Proof. By Corollary 3.3.1, A_{aug} is a DAG. By Definition 3.5, there is exactly one left leg and one right leg, or one auxiliary edge outgoing from each vertex (except s) in A_{aug}. Hence the lemma holds. □

In the following definitions and lemmas, we define $BOUN_l(f^*)$ ($BOUN_r(f^*)$) and related concepts, and describe our approach to compute $BOUN_l(f^*)$ ($BOUN_r(f^*)$). Among the sets defined below are two related and similar concepts, $BOUN_l(f^*)$ ($BOUN_r(f^*)$) and $BASE_l(f^*)$ ($BASE_r(f^*)$); these sets are different in that the latter is a superset of the former and is defined to aid the computation of the former.

DEFINITION 3.10. $PRED_l(f^*)$ ($PRED_r(f^*)$) is the set of the *proper predecessors*

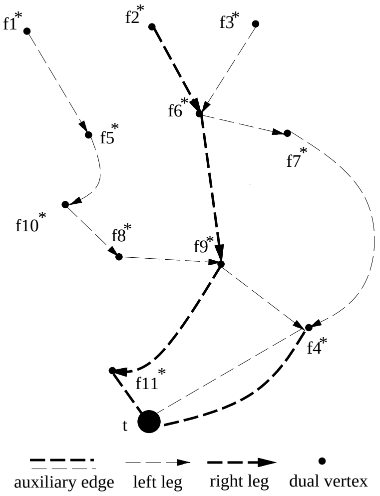

FIG. 3. *T_l^* and T_r^* for the circuit in Figure 1, given an input assignment* $(0,1,0,1)$ *to the input nodes i_1, i_2, i_3, i_4. T_l^* consists of the light dashed edges. T_r^* consists of the dark dashed edges.*

of f^* in T_l^* (T_r^*), i.e. the set of dual vertices that can reach f^* through directed paths of length at least 1 in T_l^* (T_r^*).

We associate with each dual vertex f^* in T_l^* (T_r^*) the following sets of valid bases of C_{aug}:

(i) $BASE_l(f^*)$ ($BASE_r(f^*)$) is the set of valid bases B such that the dual vertex of the left (right) bounding face of B is either f^* or in $PRED_l(f^*)$ ($PRED_r(f^*)$). (Informally, $BASE_l(f^*)$ ($BASE_r(f^*)$) is the set of the valid bases in C_{aug} whose left (right) boundaries either contain f^* or a predecessor of f^* in T_l^* (T_r^*).)

(ii) $BOUN_l(f^*)$ ($BOUN_r(f^*)$) is the set of valid bases B whose left (right) boundary contains f^*.

(iii) $JOIN(f^*) = BASE_l(f^*) \cap BASE_r(f^*)$. (Informally, $JOIN(f^*)$ is the set of valid bases whose left and right boundaries either terminate at f^*, or terminate at a predecessor of f^* and the extension of the left and right boundaries rejoin at f^*.)

(iv) $TERM(f^*)$ is the set of valid bases whose left and right boundaries terminate exactly at f^*.

For convenience, if a set for a dual vertex in A_{aug} cannot be defined through Definition 3.10 (e.g., if a dual vertex is not in T_l^*), we assume that it is empty.

Figure 4 illustrates the above definitions. For valid base $B_2 = \{i_4\}$ with left bounding face f_3 and right bounding face f_4 in Figure 1, the left boundary of B_2 is a subpath of the path from f_3^* to t in T_l^*, the right boundary of B_2 is a subpath of the path from f_4^* to t in T_r^* (in this case, the subpath is the single vertex f_4^*). Notice the difference between $BASE_l(f_4^*) = \{B_1, B_2\}$ and $BOUN_l(f_4^*) = \{B_2\}$, between $BASE_r(t) = \{B_1, B_2\}$ and $BOUN_r(t) = \phi$, and between $JOIN(t) = \{B_1, B_2\}$ and $TERM(t) = \phi$.

The relations among these sets are summarized in the following lemma. For convenience, we will focus on the sets with index l. The relations among the sets with index r are symmetric.

LEMMA 3.6. *Let f^* be a dual vertex in A_{aug}. Then*

1. $BASE_l(f_p^*) \subseteq BASE_l(f^*)$, *for any* $f_p^* \in PRED_l(f^*)$.

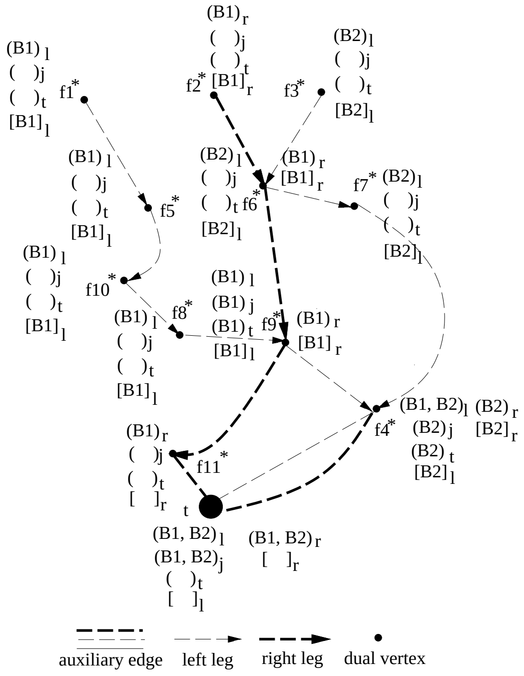

FIG. 4. *Illustrations for the sets $BASE_l$, $BASE_r$, $BOUN_l$, $BOUN_r$, $JOIN$, and $TERM$ on T_l^* and T_r^*. For each node f^* in T_l^*, the contents of $()_l$ denote $BASE_l(f^*)$, the contents of $[]_l$ denote $BOUN_l(f^*)$, the contents of $()_j$ denote $JOIN(f^*)$, and the contents of $()_t$ denote $TERM(f^*)$. For each node f^* in T_r^*, the contents of $()_r$ denote $BASE_r(f^*)$, and the contents of $[]_r$ denote $BOUN_r(f^*)$,*

2. *$BOUN_l(f^*) \cap BOUN_l(f_1^*) = \phi$ and $BASE_l(f^*) \cap BASE_l(f_1^*) = \phi$, for any f_1^* that does not have predecessor-successor relation with f^* in T_l^*.*
3. *$TERM(f^*) \cap TERM(f_1^*) = \phi$, for any $f_1^* \neq f^*$.*
4. *$TERM(f^*) = JOIN(f^*) \setminus \cup_{f_p^* \in PRED_l(f^*)} JOIN(f_p^*)$.*
5. *$BOUN_l(f^*) = BASE_l(f^*) \setminus \cup_{f_p^* \in PRED_l(f^*)} TERM(f_p^*)$; and $|BOUN_l(f^*)| = |BASE_l(f^*)| - \sum_{f_p^* \in PRED_l(f^*)} |TERM(f_p^*)|$.*

Proof. The correctness of 1, 2, 3 follows directly from Definition 3.10. The correctness of 4 follows from Definition 3.7, Definition 3.10 and 1, 2. The correctness of 5 follows from Definition 3.10 and 1, 2, 3. □

Our goal is to identify the gates that are on the left and right boundaries of some valid base. Since a left leg (f_1^*, f_2^*) is on the left boundary of some valid base iff $|BOUN_l(f_1^*)| > 0$ and $|BOUN_l(f_2^*)| > 0$, it suffices to compute $|BOUN_l(f^*)|$ for every f^* in T_l^*. $BASE_l(f^*)$ (and hence $|BASE_l(f^*)|$) can be easily computed using the Euler-tour technique [23] on T_l^* (see Procedure 2 in subsection 3.3 for details). Since $BASE_l(f^*)$ ($BASE_r(f^*)$) contains valid bases with consecutive labels (modulo the total number of bases) in the total order of the valid bases, it can be described succinctly by a range $[x, y]$ where x, y are the numbers of the first and the last valid bases in $BASE_l(f^*)$ ($BASE_r(f^*)$) respectively. If we can compute $|TERM(f^*)|$ for every f^* that is in both T_l^* and T_r^*, then we can compute $|BOUN_l(f^*)|$ using 5 in Lemma 3.6 and the Euler-tour technique. It remains to compute $|TERM(f^*)|$ for every f^* that is in both T_l^* and T_r^*.

We can try to compute $|TERM(f^*)|$ directly from 4 in Lemma 3.6. However, note that the sets $JOIN(f^*) = BASE_l(f^*) \cap BASE_r(f^*)$ are not necessarily disjoint for different f^* in T_l^* if they have predecessor-successor relation. Instead, we show in the following lemma that they satisfy some important properties, and then in Lemma 3.8 we give a formula to compute $TERM(f^*)$ with disjoint $JOIN(f^*)$ sets.

LEMMA 3.7.

1. *If f_p^* is a predecessor of f^* in both T_l^* and T_r^*, then $JOIN(f_p^*) \subseteq JOIN(f^*)$.*
2. *Otherwise, $JOIN(f^*) \cap JOIN(f_p^*) = \phi$.*

Proof.

1. By 1 in Lemma 3.6, we have both $BASE_l(f_p^*) \subseteq BASE_l(f^*)$ and $BASE_r(f_p^*) \subseteq BASE_r(f^*)$. Therefore, $JOIN(f_p^*) \subseteq JOIN(f^*)$.
2. Without lose of generality, assume f^* and f_p^* do not have predecessor-successor relation in T_l^*. Then by 2 in Lemma 3.6, $BASE_l(f^*) \cap BASE_l(f_p^*) = \phi$. Therefore, $JOIN(f_p^*) \cap JOIN(f^*) = \phi$. □

Based on Lemma 3.7, we give the following definition.

DEFINITION 3.11. For a dual vertex f^* and one of its predecessors f_p^* in T_l^* with $JOIN(f^*) \neq \phi$ and $JOIN(f_p^*) \neq \phi$, $JOIN(f_p^*)$ is *immediately enclosed* by $JOIN(f^*)$, denoted by $JOIN(f_p^*) \subseteq_I JOIN(f^*)$, *iff* $JOIN(f_p^*) \subseteq JOIN(f^*)$ and there is no dual vertex f_q^* on the directed path from f_p^* to f^* in T_l^* such that $JOIN(f_q^*) \subseteq JOIN(f^*)$.

LEMMA 3.8. *For a dual vertex f^* that is in both T_l^* and T_r^*,*

1. $TERM(f^*) = JOIN(f^*) \setminus \cup_{f_p^* \in PRED_l(f^*) \wedge JOIN(f_p^*) \subseteq_I JOIN(f^*)} JOIN(f_p^*)$,
2. $|TERM(f^*)| = |JOIN(f^*)| - \sum_{f_p^* \in PRED_l(f^*) \wedge JOIN(f_p^*) \subseteq_I JOIN(f^*)} |JOIN(f_p^*)|$.

Proof. By 4 in Lemma 3.6 and Lemma 3.7, 1 holds immediately. By Definition 3.11, none of the f_p^* in the summation in 2 has predecessor-successor relation. By Lemma 3.7, the sets $JOIN(f_p^*)$ in 2 are disjoint. Hence 2 holds. □

The above lemmas give us the necessary tools to compute $|BOUN_l(f^*)|$ efficiently in parallel. The algorithms that implement the computations in Lemmas 3.6 & 3.8 are given in procedures 2 & 3 and in the proof of Lemma 3.16 in subsection 3.3. Having computed the left and right boundaries of the valid bases of C_{aug}, our next step is to identify the regions of C_{aug} that consist of gates with value 1. In the following definition, we define a *separating graph* A_{sep}, which is a subgraph of A_{aug} that consists of the left and right boundaries of all the valid bases of C_{aug}, and which is used to find the regions of C_{aug} that consists of gates with value 1. It is formally defined as follows:

DEFINITION 3.12.

1. A *separating graph* A_{sep} contains s and the vertices f^* in A_{aug} for which either $BOUN_l(f^*) \neq \phi$ or $BOUN_r(f^*) \neq \phi$. A_{sep} contains t if $BOUN_l(t) \neq \phi$ or $BOUN_r(t) \neq \phi$.
2. An edge (f_1^*, f_2^*) of A_{aug} is an edge in A_{sep} if one of the following three conditions holds: *a*) $f_1^* = s$ and f_2 is the left or right bounding face of a valid base, or *b*) $BOUN_l(f_1^*) \neq \phi$ and $BOUN_l(f_2^*) \neq \phi$ and $BOUN_l(f_1^*) \neq TERM(f_1^*)$, or *c*) $BOUN_r(f_1^*) \neq \phi$ and $BOUN_r(f_2^*) \neq \phi$ and $BOUN_r(f_1^*) \neq TERM(f_1^*)$.

A_{sep} is a subgraph of A_{aug}, which is a subgraph of C_{aug}^* (the dual graph of C_{aug}) augmented with s and t. Hence A_{sep} is a plane graph. Each face in A_{sep} is called a *separating region* of the primal graph C_{aug}. Note that a separating region of C_{aug} either is in the internal region of a valid base, or in the external region of every valid base, in which case we call it an *external separating region*.

In the example in Figure 5, A_{sep} consists of the left and right boundaries of

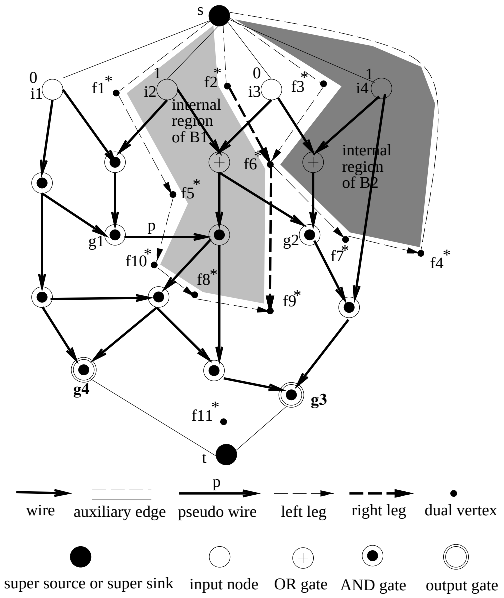

FIG. 5. *C_{aug} and A_{sep} to show the left and right boundaries of B_1 and B_2. C_{aug} consists of the solid edges. The complete input assignment to the input nodes i_1, i_2, i_3, i_4 is (0, 1, 0, 1). $B_1 = \{i_2\}$ and $B_2 = \{i_4\}$ are two valid bases. The left bounding face and the right bounding face of B_1 are f_1 and f_2 respectively. The left boundary of B_1 is the directed path $(s, f_1^*, f_5^*, f_{10}^*, f_8^*, f_9^*)$ and the right boundary of B_1 is the directed path (s, f_2^*, f_6^*, f_9^*). The part of the plane inside the two boundaries is the internal region of B_1 and the part of the plane outside the two boundaries is the external region of B_1. The left bounding face and the right bounding face of B_2 are f_3 and f_4 respectively. The left boundary of B_2 is the directed path $(s, f_3^*, f_6^*, f_7^*, f_4^*)$ and the right boundary of B_2 is the directed path (s, f_4^*). A_{sep} consists of all dashed edges.*

B_1 and B_2. The separating regions of C_{aug} in Figure 5 are: the face in A_{sep} with boundary $(s, f_1^*, f_5^*, f_{10}^*, f_8^*, f_9^*, f_6^*, f_2^*, s)$ (i.e., the internal region of B_1), the face in A_{sep} with boundary $(s, f_3^*, f_6^*, f_7^*, f_4^*, s)$ (i.e., the internal region of B_2), the face in A_{sep} with boundary $(s, f_2^*, f_6^*, f_3^*, s)$, and the face in A_{sep} with boundary $(s, f_1^*, f_5^*, f_{10}^*, f_8^*, f_9^*, f_6^*, f_7^*, f_4^*, s)$.

DEFINITION 3.13. A wire w is *incoming to (outgoing from)* a subcircuit C' if the head (tail) of w is a gate in C' but the tail (head) of w is not.

LEMMA 3.9. *All incoming wires to an external separating region R in C_{aug} are either wires with value 1 or pseudo wires. (Any input node in R will have value 0.)*

Proof. The wires incoming to R must come from the internal regions of some valid bases, since a wire crosses a boundary of a valid base B either from the internal region to the external region of B, or from the external region to the internal region of B. Since R is not in the internal region of any valid base of C_{aug}, all the wires incoming to R are wires outgoing from the internal regions of some valid bases. By Corollary 3.4.1, the gates in the internal region of any valid base of C_{aug} have value 1. Hence all incoming wires to R are either wires with value 1 or pseudo wires. ☐

Recall that Corollary 3.4.1 states that the gates in the internal region of every valid base of C_{aug} have value 1. Hence the gates in the separating regions that are in the internal region of a valid base have value 1. In the following corollary we will extend Corollary 3.4.1 to show that in fact the gates in any separating region (including external separating region) that does not contain an input node with value

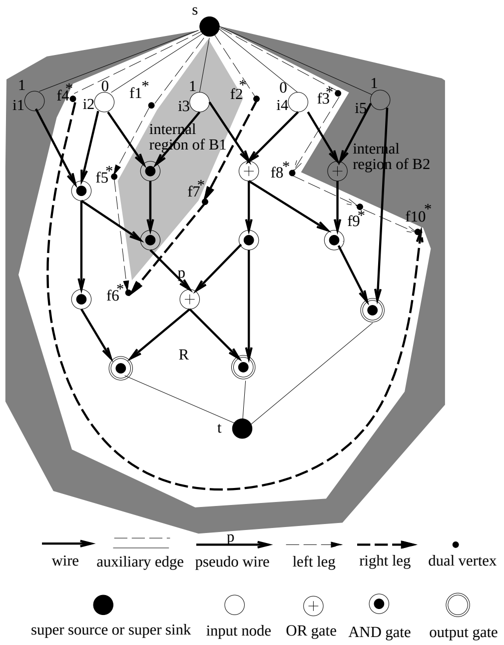

FIG. 6. *C_{aug} and A_{sep} to show the left and right boundaries of B_1 and B_2. C_{aug} consists of the solid edges. A_{sep} consists of the dashed edges. The complete input assignment to the input nodes i_1, i_2, i_3, i_4, i_5 is (1, 0, 1, 0, 1). $B_1 = \{i_3\}$ and $B_2 = \{i_5, i_1\}$ are two valid bases. The left boundary of B_1 is the directed path (s, f_1^*, f_5^*, f_6^*) and the right boundary of B_1 is the directed path (s, f_2^*, f_7^*, f_6^*). The left boundary of B_2 is the directed path $(s, f_3^*, f_8^*, f_9^*, f_{10}^*)$ and the right boundary of B_2 is the directed path (s, f_4^*, f_{10}^*).*

0 will have value 1.

COROLLARY 3.9.1. *If a separating region R of C_{aug} does not contain an input node with value 0, then all the gates of C_{aug} in R have value 1.*

Proof. If R is in the internal region of a valid base, then by Corollary 3.4.1, the lemma holds. Now we consider an external separating region R that is not in the internal region of any valid base of C. By Lemma 3.9, all incoming wires to R are either wires with value 1 or pseudo wires. Since R does not contain an input node with value 0, all the gates in R will have value 1. □

By Corollary 3.9.1, the problem of evaluating the one-input-face PMC C is now reduced to the problem of evaluating each subcircuit of C_{aug} in an external separating region that contains input nodes with value 0, since the gates in other separating regions are known to have value 1. Our next step is to transform these subcircuits into one-input-face PMCs so that we can evaluate these subcircuits recursively. One nontrivial problem with a subcircuit of C_{aug} in a separating region is that the output gates of the subcircuit may interlace with the input nodes of the subcircuit, which makes it impossible to add a super sink to the subcircuit without violating the planarity property. For example, in Figure 5, after removing the gates in the internal region of B_1 and the internal region of B_2, g_1 will be a new output gate and g_2 will be a new input gate, and the input nodes and the output gates i_1, g_1, g_2, g_3, g_4 are interlaced with each other in the resulting subcircuit. The following definition and procedure give a method we will apply to solve this problem. This construction uses pseudo wires (defined in part 5 of Definition 2.5).

Procedure 1: Subcircuit transformation
Input: C_R, the subcircuit of C_{aug} in an external separating region R containing at least one input node with value 0.
Output: C'_R, a one-input-face PMC logically equivalent to C_R.

```
0. Initialize C'_R to C_R;
1. for each wire w_i in C_aug incoming to C_R in parallel do
      let g be the gate in C_R receiving w_i as an input;
      {{note that g must be a two-input AND gate and the sink of a face}}
      (a) if the other input wire of g is in R then remove w_i from C'_R;
      (b) else (i.e., the other input wire of g is a wire incoming to C_R)
              make g a new input node with value 1 in C'_R;
          end {if};
   end {for};
2. for each wire w_o in C_aug outgoing from C_R in parallel do
      insert a one-input AND gate g_{w_o} in w_o with g_{w_o} lying inside R,
      and remove the part of w_o outgoing from g_{w_o};
      let w*_o = (f*_1, f*) be the dual edge on the boundary of R that crosses w_o;
      assume w_o is on the counterclockwise (clockwise) boundary of the face f in C_aug;
      let s_f be the sink of f in C_aug;
      let w* be the other edge connected to f* on the boundary of R in A_sep;
      (a) if w* is an outgoing edge of f*
          then {{w* crosses either the left input wire w_l or the right input wire w_r of s_f}}
              attach an output wire to g_{w_o} in C'_R by adding a pseudo wire as follows:
              {{so that g_{w_o} would not be an output gate in C'_R, and C'_R is a plane graph}}
              (i) if w* crosses w_l (w_r)
                  then {{s_f must be a two-input AND gate}}
                      connect g_{w_o} and s_f through a pseudo wire to replace w_l (w_r) in C'_R;
                  end {if};
              (ii) if w* crosses w_r (w_l)
                  then make g_{w_r} (i.e., the one-input AND gate inserted in w_r at
                      at the beginning of step 2) a two-input AND gate,
                      and connect g_{w_o} to g_{w_r} through a pseudo wire in C'_R;
                  end {if};
          end {if};
      (b) if w* is an incoming edge of f* {{f* is called a bottom of R in this case}}
          then make g_{w_o} a new output gate in C'_R;
          end {if};
   end {for};
end.
```

DEFINITION 3.14. A circuit C' is *logically equivalent* to a circuit C, if from a partially evaluated C we construct C' (possibly with additional gates) such that for each unevaluated gate g in C, there is a gate in C' with the same value as g.

The algorithm given in Procedure 1 transforms a subcircuit of C_{aug} in an external separating region that contains at least one input node with value 0 into a logically equivalent one-input-face PMC. Some examples of this transformation are given in Figure 7. We now show that C'_R constructed by Procedure 1 is a one-input-face PMC that is logically equivalent to C_R.

LEMMA 3.10. *C'_R is logically equivalent to C_R.*

Proof. We first show that step 1 in Procedure 1 does not change the value of the gates in C'_R that were originally in C_R. Since C_R is in the external region of every valid base, by Definition 3.5, a wire outgoing from the internal region of some valid base and incoming to R must be either a pseudo wire or an input wire to a two-input AND gate whose other input wire is not a pseudo wire; by Lemma 3.9, all the incoming wires to R are either wires with value 1 or pseudo wires. Hence removing a pseudo input wire or an input wire with value 1 to a two-input AND gate g (whose other input wire is not a pseudo wire) in step 1(a) will not change the value of g in

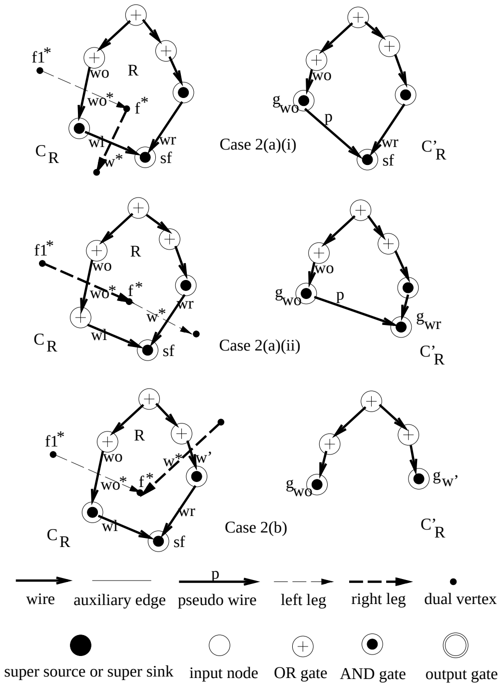

FIG. 7. *The circuit transformation of C_R to C'_R.*

C'_R; and the two-input AND gate g in step 1(b) indeed has value 1.

Step 2 in Procedure 1 reduces the number of new output gates in C'_R by adding pseudo wires. Steps 2(a)(ii) and 2(b) do not change any input to the gates of C'_R that were originally in C_R. Step 2(a)(i) changes an input to s_f by replacing w_l (w_r) with a pseudo wire. However, since w_l (w_r) is an incoming wire to the external separating region R, by the arguments given in the previous paragraph, w_l (w_r) is either a pseudo input wire or an input wire with value 1, and s_f is a two-input AND gate, and w_r (w_l) is not a pseudo wire. Hence the value of s_f depends only on the value of w_r (w_l), and is the same in both C_R and C'_R. ∎

LEMMA 3.11. *C'_R is a plane DAG.*

Proof. It is easy to see that C'_R is still a plane graph since the pseudo wires introduced in Procedure 1 will not cross any existing wires in C_R.

Suppose there is a directed cycle in C'_R. Then we map a gate g on the cycle to a gate in C_{aug} by the following function f: $f(g) = g$ if g is a gate in C_{aug}; $f(g) = g_2$ if g is not a gate in C_{aug} and g is a new gate inserted in wire (g_1, g_2) of C_{aug} in step 2 of Procedure 1. For each edge (g_1, g_2) on the cycle in C'_R, if (g_1, g_2) is not an edge in C_{aug}, then we add a new edge $(f(g_1), f(g_2))$ to C_{aug}, and call the augmented graph C'_{aug}. Hence there is a cycle in C'_{aug} containing new edges. We now prove that for each new edge $(f(g_1), f(g_2))$ in C'_{aug}, there is a directed path in C_{aug} from $f(g_1)$ to $f(g_2)$. We consider the following three cases:

(i) Case 1. Both g_1 and g_2 are gates in C_{aug}. Then $(f(g_1), f(g_2)) = (g_1, g_2)$, which is a wire in C_{aug}.

(ii) Case 2. g_1 is a newly added gate in C'_R, but g_2 is a gate in C_{aug}. Suppose g_1 is inserted in the wire (g_3, g_4) of C_{aug}. Then (g_1, g_2) is a pseudo wire added in

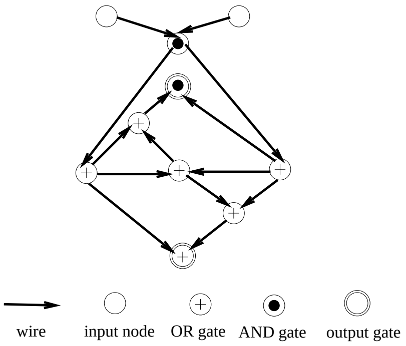

FIG. 8. *An example of a PMC with all input nodes in a single face but output gates in different faces. This PMC cannot be converted into a one-input-face PMC by adding pseudo wires to the output gates, since any pseudo wire added to an output gate will create a directed cycle in this example.*

step 2(a)(i) of Procedure 1 and g_2 is the sink of the face whose boundary contains g_4. Hence there is a directed path from $f(g_1) = g_4$ to $f(g_2) = g_2$ in C_{aug}.

(iii) Case 3. g_2 is a newly added gate in C'_R, but g_1 is a gate in C_{aug}. Then g_2 is inserted in the wire $(g_1, f(g_2))$ of C_{aug} in step 2 of Procedure 1. Hence there is a directed path from $f(g_1) = g_1$ to $f(g_2)$ in C_{aug}.

(iv) Case 4. Both g_1 and g_2 are newly added gates in C'_R. Then (g_1, g_2) is a pseudo wire added in step 2(a)(ii) of Procedure 1. Suppose g_1 is inserted in the wire (g_3, g_4) of C_{aug}, and g_2 is inserted in the wire (g_5, g_6) of C_{aug}. Then (g_5, g_6) is an outgoing edge from C_R, and g_6 is the sink of the face whose boundary contains g_4. Hence there is a directed path from $f(g_1) = g_4$ to $f(g_2) = g_6$ in C_{aug}.

Hence there is a directed cycle in C_{aug}, which contradicts the fact that C_{aug} is a DAG. Hence C'_R is acyclic. □

At this point, one might wonder if it is the case that any PMC whose input nodes are on the boundary of a single face can be converted to a one-input-face PMC by adding pseudo wires to the output gates. The example in Figure 8 shows that this is not always possible when the output gates are on the boundaries of multiple faces; the construction of C'_R exploited some special properties of a one-input-face PMC and its separating regions to guarantee that the result is a DAG, and this is not always the case when the input circuit is not a one-input-face PMC.

We now show that a subcircuit C'_R output by Procedure 1 must have all inputs in one face, and all outputs in one face, and no interlacing of inputs and outputs. This will establish that C'_R is a one-input-face PMC.

A dual vertex f^* is a *bottom* of a separating region R if it is the head of two edges (which are dual edges of C_{aug}) on the boundary of R. (See step 2(b) of Procedure 1 and case 2(b) in Figure 7.)

LEMMA 3.12. *A separating region R has at most one bottom, and if R has a bottom then R does not contain the super sink t.*

Proof. Let f^* be a bottom of R and let w_1^* and w_2^* be the two edges incoming to f^*. We find two paths P_1^* and P_2^* in A_{sep} such that (a) P_1^* goes to f^* through w_1^* and P_2^* goes to f^* through w_2^* and (b) P_1^* and P_2^* intersect with each other only at their starting vertices and their ending vertices. Let R' be the region whose counterclockwise boundary is P_1^* and whose clockwise boundary is P_2^*. Then R is inside R' since R is a face in A_{sep}.

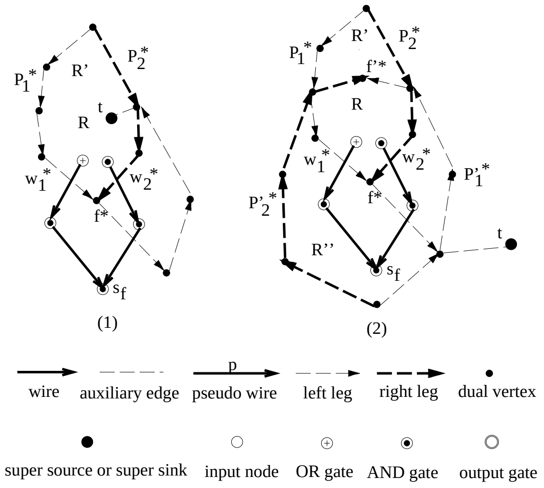

FIG. 9. *Figures for the proof of Lemma 3.12.*

We first prove that t is not in R', which implies that t is not in R (see (1) in Figure 9). Let s_f be the sink of the primal face f. If s_f is t then we have proved that t is not in R'. Otherwise, since the primal edges of w_1^* and w_2^* are outgoing from R', s_f and its two input wires must be outside of R' (note that the two input wires of s_f cannot be the primal edges of w_1^* and w_2^*, since only the dual edges outgoing from f^* can cross the input wires of s_f). Hence any outgoing edges from f^* in A_{aug} must be outside of R' since they cross the two input wires of s_f. Hence if t were in R' then A_{aug} would have contained a directed cycle, since there is a directed path from f^* to t in A_{aug}. A contradiction.

We now prove that R has at most one bottom (see (2) in Figure 9). Suppose R has another bottom $f^{'*}$. Let $w_1^{'*}$ and $w_2^{'*}$ be the two edges incoming to $f^{'*}$. We find two paths $P_1^{'*}$ and $P_2^{'*}$ in A_{sep} such that (a) $P_1^{'*}$ goes to $f^{'*}$ through $w_1^{'*}$ and $P_2^{'*}$ goes to $f^{'*}$ through $w_2^{'*}$ and (b) $P_1^{'*}$ and $P_2^{'*}$ intersect with each other only at their starting vertices and their ending vertices. Let R'' be the region whose counterclockwise boundary is $P_1^{'*}$ and whose clockwise boundary is $P_2^{'*}$. Then by the proof in the previous paragraph, t is neither in R' nor in R''. But at least one path from f^* or $f^{'*}$ to t will create a directed cycle in A_{aug}. A contradiction. ∎

COROLLARY 3.12.1. *If C_R contains a bottom, then C'_R does not contain an original output gate of C_{aug}, and C'_R contains at most two newly created output gates, and the two output gates are adjacent to each other on the boundary of a face; if C_R does not contain a bottom, then C'_R does not contain any newly created output gates (C'_R may contain some original output gates of C_{aug}).*

Proof. If C_R contains a bottom, then t is not in C'_R and hence C'_R does not contain an original output gate of C_{aug} (since the auxiliary wires connecting output gates to t do not cross the boundary of R). Further since C'_R has at most one bottom, at most two new output gates are created in C'_R and they are adjacent to each other on the boundary of a face (see case 2(b) in Figure 7). If C_R does not contain a bottom, then no new output gates are created in C'_R by the construction in Procedure 1. ∎

LEMMA 3.13. *All newly created input nodes in C'_R are on the boundary of a single face.*

Proof. After removing all the gates not in R and the wires crossing the boundary of R, all the input nodes in C'_R are on the boundary of a single face, which is the external face of C'_R. Further, the new faces created by the new pseudo wires added in step 2 of Procedure 1 do not contain input nodes on their boundaries. □

LEMMA 3.14. *Procedure 1 constructs a one-input face PMC C'_R that is logically equivalent to C_R, and it runs in $O(1)$ time using a linear number of processors on an EREW PRAM.*

Proof. Lemma 3.13 and Corollary 3.12.1 ensure that the output gates and the input nodes in C'_R do not interlace. By Lemmas 3.10, 3.11, 3.13 and Corollary 3.12.1, C'_R is a one-input-face PMC that is logically equivalent to C_R.

It is straightforward to see that all steps in Procedure 1 can be implemented in constant time using a linear number of processors. □

We conclude this subsection by showing that a subcircuit C'_R output by Procedure 1 contains at most half the number of valid bases in C_{aug}.

DEFINITION 3.15. We say two valid bases B and B' *meet* if the right boundary of B and the left boundary of B' have a common vertex. The transitive closure of the meet relation partitions the set of the valid bases in C_{aug} into equivalence classes.

LEMMA 3.15. *The number of the valid bases in C'_R is at most half of the number of valid bases in C_{aug}.*

Proof. Let g be a newly created input node in C'_R. We say g is a *descendant of* a valid base B of C_{aug} if an original input of g is in the internal region of B. This lemma follows from the following two claims.

Claim 1. Every newly created input node in C'_R is a descendant of at least two distinct valid bases of C_{aug}, and the two valid bases are in the same equivalence class of C_{aug}.

Claim 2. The newly created input nodes in C'_R that are descendants of the valid bases in the same equivalence class of C_{aug}, are in the same valid base in C'_R.

By Claim 1, a singleton equivalence class of C_{aug} does not generate a new input node with value 1 in C'_R. By Claim 2, an equivalence class of C_{aug} containing at least two valid bases generates at most one valid base in C'_R. Hence the lemma holds.

We first prove Claim 1. Since only the sink of a face can have its input wires crossed by the dual edges in A_{sep}, a newly created input node g must be the sink of a face f in C_{aug}. The two original input wires w_1 and w_2 of g must cross a dual edge w_1^* (which is a left leg) on the left boundary of a valid base and a dual edge w_2^* (which is a right leg) on the right boundary of a valid base respectively. Since w_1^* and w_2^* are outgoing from the same vertex f^*, they cannot be on the left and right boundaries of the same valid base (since the left and right boundaries end at their first common vertex after s). Therefore, the input wires of g are outgoing from the internal regions of at least two different valid bases (the internal regions of several valid bases may overlap). Further, the two valid bases are in the same equivalence class since f^* is a common vertex of the left boundary of one valid base and the right boundary of the other valid base.

We now prove Claim 2. Since the external separating region R contains at least one input node with value 0, the boundary of R must contain the super source s. Further, s may appear on the boundary of R more than once (see Figure 6, s appears twice on the boundary of the external separating region R which consists of the part of the plane between the internal region of B_1 and the internal region of B_2). Since multiple appearances of s is possible, if we remove s from the boundary of R, the boundary will be divided into several connected portions, each enclosing a disjoint part of C_{aug} (in Figure 6, the two disjoint parts are the internal region of B_1 and the internal region of B_2). The valid bases in one part cannot be in the same equivalence

Algorithm 1: Complete evaluation of a one-input-face PMC
Input: An embedded one-input-face PMC C and a complete input assignment to C.
Output: Each gate in C is assigned a value 0 or 1.
1. **if** all input nodes in C have value 1
2. **then** assign value 1 to all gates in C; **return**;
   else if all input nodes in C have value 0
3. **then** assign value 0 to all gates in C; **return**;
   end {**if**};
   end {**if**};
4. Augment C to C_{aug}, and construct the auxiliary dual graph A_{aug};
5. Find the edges in A_{aug} that are on the boundaries of valid bases of C_{aug} (see Procedure 2);
6. Construct the separating graph A_{sep};
7. Remove the wires in C_{aug} that cross the boundary edges of A_{sep};
8. Find the (undirected) connected components in the remaining C_{aug};
9. **for** each connected component C_R found in step 8 **in parallel do**
10. **if** C_R does not contain input nodes with value 0
11. **then** assign value 1 to all gates in C_R;
12. **else** transform C_R to C'_R using Procedure 1;
13. Recursively evaluate C'_R;
    end {**if**};
   end {**for**};
end.

Procedure 2: Finding the edges in A_{aug} that are on the boundaries of valid bases
Input: C_{aug}, A_{aug}, and a complete input assignment to C_{aug}.
Output: The edges of A_{aug} that are on the boundaries of valid bases of C_{aug} are marked.
2.1. Find all the valid bases in C_{aug}, and label them in the order of the sequence in which they appear on the boundary of the input face of C_{aug};
2.2. Construct T_l^* and T_r^* from A_{aug};
2.3. Compute $BASE_l(f^*)$ and $BASE_r(f^*)$ for each dual vertex f^* in A_{aug};
2.4. Compute $JOIN(f^*) = BASE_l(f^*) \cap BASE_r(f^*)$ for each dual vertex f^* in A_{aug};
2.5. Find the enclosure relation $\subseteq_I$ among the $JOIN(f^*)$ (see Procedure 3);
2.6. Compute $|TERM_l(f^*)|$ and $|TERM_r(f^*)|$ for each dual vertex f^* in A_{aug} using Lem. 3.8;
2.7. Compute $|BOUN_l(f^*)|$ and $|BOUN_r(f^*)|$ for each dual vertex f^* in A_{aug} using Lem. 3.6;
2.8. Mark all dual edges (f^*,g^*) in A_{aug} with $|BOUN_l(f^*)| > 0$ and $|BOUN_l(f^*)| > |TERM(f^*)|$, or with $|BOUN_r(f^*)| > 0$ and $|BOUN_r(f^*)| > |TERM(f^*)|$;
end.

class as a valid base in a different part. Let P be a connected portion of the boundary of R after removing S. Let I be the set of all newly created input nodes that are descendants of the valid bases in the equivalence classes enclosed in P. Then the original input wires of the input nodes in I must cross the dual edges in P. Hence the input nodes in I are adjacent on the boundary of a face in C'_R, and therefore are in the same valid base in C'_R. □

3.3. An Efficient Algorithm for the One-Input-Face PMCVP. Based on the approach we presented in the previous subsection, we give an efficient EREW PRAM algorithm, called Algorithm 1, for evaluating a one-input-face PMC.

The correctness and complexity analysis of Algorithm 1 will be given in Theorem 3.1. All steps in Algorithm 1 are quite straightforward to implement except step 5, which is implemented by Procedure 2. Step 2.5 in Procedure 2 is implemented by Procedure 3, which is similar to a procedure used for the layered PMC in [20].

LEMMA 3.16. *Procedure 2 (i.e., step 5 in Algorithm 1) correctly finds the edges in A_{aug} that are on the boundaries of valid bases of C_{aug}, and it runs in $O(\log n)$ time using a linear number of processors on an EREW PRAM.*

Proof. The correctness of all steps (except step 2.5) of Procedure 2 which implements step 5 of Algorithm 1, has been proved in Lemmas 3.6 and 3.8. We now show

Procedure 3: Finding the enclosure relation $\subseteq_I$ for the $JOIN(f^*)$
Input: T_l^* and $JOIN(f^*)$ for each dual vertex f^* on T_l^*.
Output: The enclosure forest EF^* such that a dual vertex f_p^* is the immediate predecessor of a vertex f^* in EF^* iff $JOIN(f_p^*) \subseteq_I JOIN(f^*)$.

3.1. **for** each vertex f^* with nonempty $JOIN(f^*)$ and with the length of the longest path from a leaf to f^* in T_l^* being k **in parallel do**
3.2. Assign two triples $(x,\ -k,\ f^*)$, $(y,\ k,\ f^*)$ for each range $[x, y]$ in $JOIN(f^*)$;
end {**for**};
3.3. Sort all triples into nondecreasing order according to the first two elements in a triple;
3.4. **for** each triple $(x,\ -k,\ f^*)$, where $k \geq 0$ **in parallel do**
3.5. Find its previous triple $(n',\ k',\ f'^*)$ in the sorted list;
3.6. **if** $(k' < 0)$ and $(f^* \neq f'^*)$ **then** f'^* is the parent of f^* in EF^*;
3.7. **else if** $(k' > 0)$ and $(f^* \neq f'^*)$ **then** f'^* is the left sibling of f^* in EF^*; **end** {**if**};
end {**if**};
end {**for**};
3.8. Construct the EF^* from the parent and sibling relations;
end.

the correctness of Procedure 3, which implements step 2.5 of Procedure 2. Let f_1^* and f_2^* be two vertices in T_l^* such that the longest paths from a leaf to f_1^* and from a leaf to f_2^* are of length k_1 and k_2 respectively. By Lemma 3.7, f_1^* is a successor of f_2^* in T_l^* and $JOIN(f_1^*) \supseteq JOIN(f_2^*)$ *iff* $k_1 > k_2$, and for each range $[x_2, y_2]$ of $JOIN(f_2^*)$ there exists a range $[x_1, y_1]$ of $JOIN(f_1^*)$, such that $x_1 \leq x_2 \leq y_2 \leq y_1$ in the cyclic order; $JOIN(f_1^*) \cap JOIN(f_2^*) = \phi$ *iff* for each range $[x_2, y_2]$ of $JOIN(f_2^*)$ and each range $[x_1, y_1]$ of $JOIN(f_1^*)$, $x_1 \leq y_1 < x_2 \leq y_2$ in the cyclic order. Therefore, if (n', k', f'^*) and $(x, -k, f^*)$ are two consecutive triples in the sorted list, we have: (1) if $k' < 0$ and $f^* \neq f'^*$ then $JOIN(f^*) \subseteq_I JOIN(f'^*)$ and f'^* must be the immediate successor of f^* in the EF^*; (2) if $k' > 0$ and $f^* \neq f'^*$ then f^* and f'^* share the common immediate successor in EF^*.

Next, we analyze the time complexity of Procedure 2.

In step 2.2, T_l^* (T_r^*) can be computed using Euler-tour technique as follows. We first remove s and all right (left) legs from A_{aug}. Then the resulting graph A'_{aug} is a tree rooted at t, by the uniqueness of left (right) legs. We then mark the leaf nodes of A'_{aug} that are the dual vertices of the left (right) bounding faces of valid bases of C. Finally we apply the Euler-tour technique to find all the successors of the marked leaf nodes, and the resulting subtree of A'_{aug} is T_l^* (T_r^*).

In step 2.3, since $BASE_l(f^*)$ ($BASE_r(f^*)$) contains valid bases with consecutive labels (modulo the total number of bases) in the total order of the valid bases, it can be described succinctly by a range $[l,\ h]$ where l, h are the numbers of the first and the last valid bases in $BASE_l(f^*)$ ($BASE_r(f^*)$) respectively. $BASE_l(f^*)$ ($BASE_r(f^*)$) can be computed using Euler-tour technique on T_l^* (T_r^*) as follows. We first label each leaf node of T_l^* (T_r^*) with the label of its corresponding valid base. Then for each vertex f^* in T_l^* (T_r^*), we apply the Euler-tour technique to find the smallest label and the largest label among the leaf predecessors of f^* in T_l^* (T_r^*), and assign them to l and h respectively.

In step 2.4, $JOIN(f^*)$ can be computed from $BASE_l(f^*)$ and $BASE_r(f^*)$ in constant time and be represented by at most two ranges.

Based on the above analysis, we conclude that steps 2.2-2.4 can be implemented in $O(\log n)$ time using a linear number of processors.

Procedure 3 (which implements step 2.5 in Procedure 2) can be implemented in $O(\log n)$ time with a linear number of processors using the parallel merge sort of [2] and Euler-tour technique.

Algorithm 2: Partial evaluation of a one-input-face PMC
Input: A one-input-face PMC C and a partial input assignment to C.
Output: Each gate in C that can be evaluated is assigned a value 0 or 1.
1. Assign value 1 to all input nodes with unknown value in C and apply Algorithm 1;
2. Let A be the set of the gates assigned value 0 in this solution of step 1;
3. Assign value 0 to all input nodes with unknown value in C and apply Algorithm 1;
4. Let B be the set of the gates assigned value 1 in this solution of step 3;
5. Assign value 0 to all gates in A, assign value 1 to all gates in B, and assign unknown value to the gates of C that are neither in A nor in B;

end.

It is easy to see that all other steps of Procedure 2 can be implemented in $O(\log n)$ time using a linear number of processors by computing prefix sums and applying Euler-tour and tree evaluation techniques. ∎

THEOREM 3.1. *Algorithm 1 correctly solves the complete evaluation problem of a one-input-face PMC given a complete input assignment, and it runs in $O(\log^2 n)$ time using n processors on an EREW PRAM, where n is the size of the circuit.*

Proof. Steps 1-4 are quite straightforward. The correctness of step 5 is proved by Lemma 3.16. The correctness of steps 6-13 are proved by Corollary 3.9.1 and Lemma 3.14.

It is straightforward to see that all steps except steps 5, 8, and 13 in Algorithm 1 can be implemented in $O(\log n)$ time using a linear number of processors. Lemma 3.16 shows that step 5 can be implemented in the same time complexity. Step 8 can be implemented in $O(\log n)$ time optimally by applying the algorithm in [5] for finding connected components in a planar undirected graph.

By Lemma 3.15, the number of the recursive levels needed to complete the evaluation is $O(\log n)$. Therefore, the overall time needed by Algorithm 1 is bounded by $O(\log^2 n)$. Further, the total number of gates in all remaining subcircuits C_R in step 12 in Algorithm 1 is less than the number of gates in the original C_{aug} since for each newly inserted gate in C_R, there is a unique gate in the internal region of a valid base being removed. Therefore, the processor bound holds. ∎

3.4. Partial Evaluation of a One-Input-Face PMC. We extend Algorithm 1 to solve the partial evaluation problem of a one-input-face PMC in Algorithm 2.

THEOREM 3.2. *Algorithm 2 correctly solves the partial evaluation problem of a one-input-face PMC given a partial input assignment, and it runs in $O(\log^2 n)$ time using n processors on an EREW PRAM, where n is the size of the circuit.*

Proof. By the monotonicity of the circuit, A is a subset of the gates that should be evaluated to 0 in the partial evaluation of C, and B is a subset of the gates that should be evaluated to 1 in the partial evaluation of C. Further, we now show that a gate g of C that is neither in A nor in B should have unknown values in the partial evaluation of C. Suppose not. Let g be a gate that should be evaluated to 0 (1) in the partial evaluation of C and let g be in neither A nor B. Then g evaluates to 0 (1) under every possible input assignment to the input nodes with unknown values in C. In particular, g has value 0 (1) when all input nodes with unknown values are assigned value 1 (0), which means g is in A (B). A contradiction. Therefore, Algorithm 2 is correct.

It is easy to see that the time complexity of Algorithm 2 is the same as that of Algorithm 1 since it is dominated by the two calls on Algorithm 1. ∎

4. The Face Induced PMC. In this section, we consider a face f induced circuit C_f, which is defined in Section 2. For convenience, we assume that C_f is

Algorithm 3: f-partial evaluation of a face f induced circuit C_f
Input: A face f induced circuit C_f with an f-partial input assignment.
Output: The solution of the f-partial evaluation problem of C_f.
1. **if** C_f contains only one gate **then return** the value of the gate **end** {**if**};
2. Obtain a topological ordering of the gates in C_f;
3. Let m be the total number of the non-input gates in C_f;
4. Find g_1 such that there are $\lfloor m/2 \rfloor$ non-input gates before g_1 in the topological ordering;
5. Partition the gates in C_f into two parts P_l and P_h, such that P_l contains g_1 and the gates before g_1 in the ordering, and P_h contains the gates after g_1 in the ordering, and remove the wires of C_f pointing from gates in P_l to gates in P_h;
6. **for** each gate g in P_h **in parallel do**
    if all input wire(s) of g are removed
    then replace g by an input node with unknown value in P_h;
    else if g is a two-input gate and only one input wire of g is removed
        then add an input node i with unknown value and a wire from i to g in P_h;
        end {**if**};
    end {**if**};
  end {**for**};
7. Find the (undirected) connected subcircuits in P_l and P_h;
8. f'-partially evaluate every connected subcircuit in P_l and P_h recursively in parallel, where f' is the external input face of the subcircuit;
  {{it will be shown below that each such subcircuit is a face f' induced circuit with an f'-partial assignment}}
9. Remove all gates that are assigned 0 or 1 in step 8 in P_h;
10. Assign the output values of P_l to the input nodes of P_h;
11. Partially evaluate every connected subcircuit in P_h using Algorithm 2 in parallel;
  {{it will be shown below that each such subcircuit is a one-input-face PMCs}}

end.

embedded with f being the external face. An *f-partial input assignment* to C_f is a partial input assignment where only input nodes in f can have unknown values, and the input nodes in faces other than f must have values 0 or 1. The problem of partially evaluating C_f given an f-partial input assignment is called the *f-partial evaluation* of C_f. Algorithm 3 gives our method to perform an f-partial evaluation of C_f. Algorithm 3 is similar to an algorithm in [3] which first layers a face induced circuit (which squares the size of the circuit) and then recursively partitions the circuit at an appropriate layer. Our algorithm performs a more efficient evaluation by working on a face induced circuit directly and partitioning the circuit according to its topological ordering. It then partially evaluates each subcircuit either recursively or using Algorithm 2.

Recall that a *topological ordering* of a digraph is a linear ordering of its vertices such that every edge in the graph points from a lower-numbered vertex to a higher-numbered vertex. It is well known that a digraph has a topological ordering iff it is a DAG. We now prove the correctness of Algorithm 3 and analyze its complexity.

LEMMA 4.1. *Immediately before step 8, every connected subcircuit in P_l and P_h is a face f' induced circuit for some face f', with an f'-partial input assignment.*

Proof. Let us add to C_f a super source s in face f and a super sink t in the output face of C_f for the purpose of the proof. We connect s to each input node in f with an edge, and connect each output gate to t with an edge. The resulting C_f is still a plane graph.

Only input nodes in f can have unknown values in each connected subcircuit in P_l, since no new input nodes are created in P_l. We now show that the output gates in P_l are in the same face. By step 5, every directed path from a gate in P_h to t consists only of gates in P_h. Hence the gates in P_h can be coalesced to t and the resulting

C_f is still a plane graph. The wires outgoing from gates in P_l to gates in P_h are now incoming to t. Hence after we cut the wires outgoing from gates in P_l to t and remove t, the output gates of the connected subcircuits in P_l are in a single face, which we call f_1. Hence every connected subcircuit in P_l is still a face f induced circuit with an f-partial input assignment.

P_h is $C \setminus P_l$ plus some new input nodes with unknown values generated in step 6. The output gates in P_h are not changed and hence are still in the same face. The new input nodes with unknown values are in the same face f_1, since all gates in P_l can be coalesced to s. If there are original input nodes in f remaining in P_h (which are the only input nodes in C_f that possibly carry unknown value) then f_1 must be identical to f. Hence every connected subcircuit in P_h is still a face f_1 induced circuit with an f_1-partial input assignment. □

LEMMA 4.2. *Immediately before step 11, every connected subcircuit in P_h is a one-input-face PMC.*

Proof. We show that after removing all gates assigned 0 or 1 in P_h in step 9, no new input nodes are generated, i.e., no gate with in-degree 1 or 2 in P_h becomes a gate with in-degree 0. Let g be a gate with in-degree at least 1 in P_h just before step 9. If all gate(s) that provide inputs to g have known values, then the value of g should be evaluated in step 8 and g should be removed in step 9. If all gate(s) that provide inputs to g have unknown values, then the in-degree of g is not changed. If one input of a two-input gate g has unknown value and the other has known value, then the in-degree of g is 1 after step 9. Hence no new input nodes are generated in P_h in step 9. By Lemma 4.1, every connected subcircuit in P_h and P_l in step 8 is a face f' induced circuit for some input face f' with an f' partial input assignment. Therefore, immediately before step 11, the only input nodes left in each connected subcircuit in P_h are the input nodes in f' that carry unknown value. Hence every connected subcircuit in P_h is a one-input-face PMC. □

THEOREM 4.1. *Algorithm 3 correctly solves the f-partial evaluation problem of a face f induced circuit C_f, and it runs in $O(\log^4 n)$ time using n processors on an EREW PRAM, where n is the size of C_f.*

Proof. The correctness of steps 8 & 11 are shown by Lemma 4.1 and Lemma 4.2. It is straightforward to see that other steps in Algorithm 3 are correct.

Step 1 takes constant time. Step 2 can be implemented in $O(\log^3 n)$ time using n processors on an EREW by Theorem 4.1 in Kao & Klein [12]. The connectivity of a plane undirected graph in steps 8 & 11 can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW by the algorithm in Gazit [5]. Steps 3-6 & 9-10 can be implemented in $O(\log n)$ time using $n/\log n$ processors. Step 11 takes $O(\log^2 n)$ time using n processors by Theorem 3.2. Let n' be the number of non-input gates in the original C_f. Since the in-degree of each gate in C_f is ≤ 2, we have $n' < n \leq 3n'$. Each of P_h and P_l contains at most $\lceil n'/2 \rceil$ non-input gates, and therefore at most $3\lceil n'/2 \rceil$ total gates (including the new input nodes). Let $T(n)$ be the time needed for Algorithm 3 to partially evaluate a circuit with n gates. We have

$$T(3n') \leq T(3\lceil n'/2 \rceil) + O(\log^3 n).$$

Solving the above recurrence equation, we have $T(3n') = O(\log^4 n)$. Hence $T(n) \leq T(3n') = O(\log^4 n)$. □

5. The General PMCVP. In this section, we give in Algorithm 4 our overall algorithm for evaluating a general PMC. This algorithm evaluates a general PMC recursively by decomposing it into smaller PMCs and disjoint face induced subcircuits.

Algorithm 4: Complete evaluation of a general PMC
Input: A general PMC C with input nodes $i_1, \ldots, i_m$, and a complete input assignment.
Output: Each gate in C is assigned a value 0 or 1.
0. **if** C contains only one gate **then return** the value of the gate **end** {**if**};
1. Find the smallest k, $0 \leq k \leq m$, such that every connected subcircuit in
   $C \setminus Reach(i_1, i_2, \ldots, i_{(k+1)})$ is of size $\leq n/2$ (see Figure 10);
   Let P be a connected subcircuit of size $> n/2$ in $C \setminus Reach(i_1, i_2, \ldots, i_k)$ when $k \geq 1$;
2. **if** $k \geq 1$
3. **then** Recursively solve the complete evaluation problem for the connected subcircuits in
   $C \setminus Reach(P)$ and in $P \setminus Reach(i_{(k+1)})$ (whose sizes are all $\leq n/2$) in parallel
   (see Figure 10 and Lemmas 5.1 & 5.2 for steps 3-8);
   {{it will be shown that each such subcircuit is a general PMC with a complete input assignment}}
4. Completely evaluate $Induced(i_{(k+1)}) \cap P$ using Algorithm 3;
   {{it will be shown that each such subcircuit is a face induced circuit with a complete input assignment}}
   {{now all gates in P and $C \setminus Reach(P)$ are completely evaluated}}
5. Remove P from C, let $o_1, \ldots, o_{m'}$ be the gates of P with wires outgoing to $reach(P)$;
   {{$o_1, \ldots, o_{m'}$ are on the boundary of a single face in $reach(P)$}}
6. Completely evaluate $Induced(o_1, \ldots, o_{m'})$ (i.e. $Reach(P) \setminus P$) using Algorithm 3;
   {{it will be shown that each such subcircuit is a face induced circuit with a complete input assignment}}
7. **else** Recursively solve the complete evaluation problem for the connected subcircuits in
   $C \setminus Reach(i_1)$ (whose sizes are all $\leq n/2$) in parallel;
8. Evaluate $Induced(i_1)$ using Algorithm 3;
   end {**if**};
end.

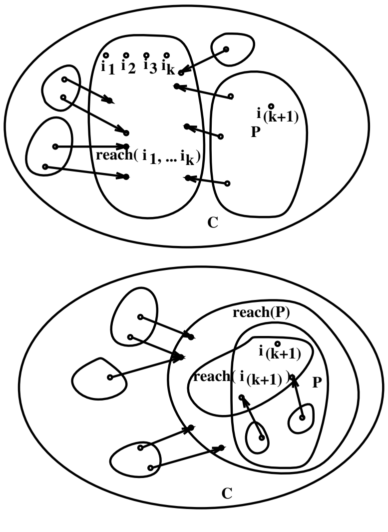

FIG. 10. *A general PMC C of size n, where P is a connected subcircuit of size $> n/2$ in $C \setminus reach(i_1, i_2, \ldots, i_k)$, but $C \setminus reach(i_1, i_2, \ldots, i_{(k+1)})$ does not contain any connected subcircuit of size $> n/2$.*

The smaller PMCs are evaluated recursively while each face induced subcircuit is evaluated by Algorithm 3. We then show the correctness and complexity of Algorithm 4 in Lemma 5.1 and Theorem 5.1. A sketch of an algorithm similar to Algorithm 4 is given in [3].

LEMMA 5.1. *Each connected subcircuit in steps 3 & 7 is a general PMC with a complete input assignment.*

Proof. Since the gates in $Reach(P)$ can be coalesced into a single gate, the output gates in $C \setminus Reach(P)$ are in the same face. Similarly the output gates in P and $P \setminus Reach(i_{(k+1)})$ are in the same face. The input nodes in $C \setminus Reach(P)$ are original input nodes in C. Since there is no wire in C outgoing from a gate in $C \setminus P$ to P, the input nodes in $P \setminus Reach(i_{(k+1)})$ are also original input nodes in C. Hence each connected subcircuit in $C \setminus Reach(P)$ and $P \setminus Reach(i_{(k+1)})$ is a general PMC with a complete input assignment, and can be completely evaluated recursively in step 3. Similar proof holds for step 7. □

LEMMA 5.2. *Each connected subcircuit in steps 4, 6 & 8 is a face induced circuit with a complete input assignment.*

Proof. The output gates in $Induced(i_{(k+1)}) \cap P$ are in the same face since they are a subset of the output gates in P. $Induced(i_{(k+1)}) \cap P$ are reachable from the original input $i_{(k+1)}$. The other new input nodes in $Induced(i_{(k+1)}) \cap P$ get their value from $P \setminus Reach(i_{(k+1)})$, which is completely evaluated in step 3. Hence $Induced(i_{(k+1)}) \cap P$ is a face f (that contains $i_{(k+1)}$) induced circuit with a complete input assignment, and can be completely evaluated using Algorithm 3 in step 4.

The output gates in $Induced(o_1, \ldots, o_{m'})$ (i.e. $Reach(P) \setminus P$) are the output gates in $Reach(P)$, and the output gates in $Reach(P)$ are a subset of the output gates in C and are in the same face. The input nodes $o_1, \ldots, o_{m'}$ in $Reach(P) \setminus P$ are the output gates in P, and are in the same face, which we call f_1, and are completely evaluated in steps 3 & 4. All gates in $Reach(P) \setminus P$ are reachable from the input nodes $o_1, \ldots, o_{m'}$ in f_1. The other input gates in $Reach(P) \setminus P$ get values from gates in $C \setminus Reach(P)$, which is completely evaluated in step 3. Hence $Induced(o_1, \ldots, o_{m'})$ (i.e. $Reach(P) \setminus P$) is a face f_1 induced circuit with a complete input assignment, and can be completely evaluated using Algorithm 3 in step 6.

Similar proof holds for step 8. □

THEOREM 5.1. *Algorithm 4 correctly solves the PMCVP for a general PMC C, and it runs in $O(\log^6 n)$ time using n processors on an EREW PRAM, where n is the size of the circuit.*

Proof. The correctness of Algorithm 4 has been shown in Lemmas 5.1 & 5.2.

The reachability in steps 1, 3, and 7 can be implemented in $O(\log^4 n)$ time using n processors on an EREW by the multiple-source reachability algorithm for planar digraphs in Guattery & Miller [10]. The k in step 1 can be found by a binary search. Hence the total time needed in step 1 is $O(\log^5 n)$. The connectivity of a plane undirected graph in steps 1, 3, and 7 can be solved in $O(\log n)$ time using n processors on an EREW by the algorithm in Gazit [5]. By Theorem 4.1, steps 4-6 & 8 can be implemented in $O(\log^4 n)$ time using n processors on an EREW. It is easy to see that the connected subcircuits in steps 3 & 7 are of size $\leq n/2$, and the subcircuits obtained in each step are disjoint. Let $T(n)$ be the time needed for Algorithm 4 to evaluate a PMC with n gates. We have

$$T(n) = T(n/2) + O(log^5 n).$$

Solving the above recurrence equation, we have $T(n) = O(\log^6 n)$. □

Note that the high power in the logarithm for the running time is mainly due to the running time of the reachability algorithms in [10] and [12]. An improvement in the running time of the parallel algorithms for reachability in a plane DAG would imply an improvement in the running time of our algorithm.

REFERENCES

[1] A. Borodin, *On relating time and space to size and depth*, SIAM J. Comput., 6 (1977), pp. 733–744.

[2] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.

[3] A. L. Delcher and S. R. Kosaraju, *An NC algorithm for evaluating monotone planar circuits*, SIAM J. Comput., to appear.

[4] P. W. Dymond and S. A. Cook, *Hardware complexity and parallel computation*, Proc. 21th IEEE Symp. on Foundations of Comp. Sci., 1980, pp. 360–372.

[5] H. Gazit, *An optimal deterministic* EREW *parallel algorithm for finding connected components in a low genus graph*, Proc. 5th International Parallel Processing Symp., 1991, pp. 84–90.

[6] A. M. Gibbons and W. Rytter, *An optimal parallel algorithm for dynamic expression evaluation and its applications*, Symp. on Foundations of Software Technology and Theoretical Comp. Sci., Springer-Verlag, 1986, pp. 453–469.

[7] L. M. Goldschlager, *A space efficient algorithm for the monotone planar circuit value problem*, Information Processing Letters, 10 (1980), pp. 25–27.

[8] L. M. Goldschlager, *A unified approach to models of synchronous parallel machines*, Proc. 10th ACM Symp. on Theory of Comput., 1978, pp. 89–94.

[9] L. M. Goldschlager, *The monotone and planar circuit value problems are* log *space complete for P"*, SIGACT News, 9 (1977), pp. 25–29.

[10] S. Guattery and G. L. Miller, *A contraction procedure for planar directed graphs*, Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, 1992, pp. 431–441.

[11] M. D. Hutton and A. Lubiw, *Upward planar drawing of single source acyclic digraphs*, Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, 1991, pp. 203–211.

[12] M. Y. Kao and P. Klein, *Toward overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs*, Proc. 22nd ACM Symp. on Theory of Comput., 1990, pp. 181–192.

[13] M. Y. Kao and G. Shannon, *Local reorientation, global order, and planar topology*, Proc. 18th ACM Symp. on Theory of Comput., 1986, pp. 160–168.

[14] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared memory machines*, Handbook of Theoretical Computer Science, J. Van Leeuwen, ed., North Holland, 1990, pp. 869–941.

[15] S. R. Kosaraju and A. L. Delcher, *Optimal parallel evaluation of tree-structured computations by raking*, Proc. 3rd Aegean Workshop on Comput., Springer-Verlag LNCS 319 (1988), pp. 101–110.

[16] R. E. Ladner, *The circuit value problem is* log *space complete for P*, SIGACT News, 1975, pp. 18–20.

[17] E. M. Mayr, *The dynamic tree expression problem*, Proc. Princeton Workshop on Algorithms, Architecture and Technology Issues for Models of Concurrent Computation, Chap. 10, 1987, pp. 157–179.

[18] G. L. Miller, V. Ramachandran and E. Kaltofen, *Efficient parallel evaluation of straight-line code and arithmetic circuits*, SIAM J. Comput., 17 (1988), pp. 687–695.

[19] V. Ramachandran and J. H. Reif, *Planarity testing in parallel* Technical Report, TR 90-15, Dept. of Computer Science, Univ. of Texas at Austin, 1990; Preliminary version appears as *An optimal parallel algorithm for graph planarity*, Proc. 30th IEEE Symp. on Foundations of Comp. Sci., 1989, pp. 282–287.

[20] V. Ramachandran and H. Yang, *An efficient parallel algorithm for the layered planar monotone circuit value problem*, Proc. 1st European Symp. on Algorithms, Springer-Verlag, LNCS 726, Bad Honnef, Germany, 1993, pp. 321–332.

[21] V. Ramachandran and H. Yang, *An efficient parallel algorithm for the general planar monotone circuit value problem*, Proc. 5nd ACM-SIAM Symp. on Discrete Algorithms, 1994, pp. 622–631.

[22] V. Ramachandran and H. Yang, *Finding the closed partition of a planar graph*, Algorithmica, 11 (1994), pp. 443–468.

[23] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.

[24] H. Yang, *An NC algorithm for the general planar monotone circuit value problem*, Proc. 3rd IEEE Symp. on Parallel and Distributed Processing, 1991, pp. 196–203.