Local Unidirectional Bias for Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning

Andrew B. Kahng, *Member, IEEE,* and Xu Xu

***Abstract*—Traditional multilevel partitioning approaches have shown good performance with respect to cutsize, but offer no guarantees with respect to system performance. Timing-driven partitioning methods based on iterated net reweighting, partitioning, and timing analysis have been proposed (Ababei *et al.*, 2002), as well as methods that apply degrees of freedom such as retiming (Cong *et al.*, 2000), (Cong *et al.*, 2002). In this paper, we identify and validate a simple approach to timing-driven partitioning based on the concept of "V-shaped nodes." We observe that the presence of V-shaped nodes can badly impact circuit performance, as measured by maximum hopcount across the cutline or similar path delay criteria. We extend traditional the Fiduccia–Mattheyses (FM) variant of the Kernighan–Lin (Kernighan and Lin, 1970) algorithm approaches to directly eliminate or minimize "distance-k V-shaped nodes" in the bipartitioning solution, achieving an attractive tradeoff between cutsize and path delay. Experiments show that in comparison to MLPart (Caldwell *et al.*, 2000), our method can reduce the maximum hopcount by 39% while only slightly increasing cutsize and runtime. No previous method improves path delay in such a transparent manner. The new partitioner is incorporated into a placer (http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/Capo/) and circuit delay is evaluated by a commercial static timing analyzer (http://www.ece.uci.edu/eceware/cadence_docs/pearluser/). The empirical results show that the delay is significantly reduced, at the cost of very acceptable impacts on wirelength and runtime.**

***Index Terms*—Fiduccia–Mattheyses (FM), hypergraph bipartitioning, timing, very large scale integration (VLSI).**

I. Introduction

With increased system complexity, circuit hypergraph partitioning, which is the "divide" step of the divide-and-conquer paradigm, plays a crucial role in many design tasks [8]. The problem is dividing the nodes of a graph into several roughly equal parts; the traditional objective is to minimize cutsize. Among many partitioning methods, multilevel approaches (e.g., MLPart [3] or hMetis [15]) are considered effective for cutsize minimization. Such methods perform a sequence of netlist coarsening and uncoarsening steps with FM-based partitioning refinement at each level of the uncoarsening hierarchy. While these algorithms outperform other algorithms in cutsize, they cannot guarantee production of small delay in general.

For performance-driven contexts, the hypergraph partitioner must consider the impact of implied interconnects[1] on performance. The primary objective of a *performance-driven partitioner* is to minimize path delay on timing paths. Recently, several performance-driven partitioning methods have been proposed. Most of these methods do not consider cutsize, and no attractive cutsize-delay tradeoff (let alone transparent consideration of path delay within traditional cutsize-driven approaches) has been discovered. In particular, existing performance-driven partitioners either try to modify the input of multilevel FM partitioners, by means such as reweighting [2], or else apply novel approaches such as min-delay clustering [9]. However, these approaches may be impractical because of large cutsize [7] or large runtime [2]. Furthermore, improvements in timing are often not obvious. The goal of our work is to find a performance-driven partitioner that can provide a more attractive, and hopefully tunable, cutsize/delay tradeoff. Our contribution is summarized as follows.

1) We define the concept of a *V-shaped node* in a partitioning solution, as well as its generalization to *distance-k V-shaped nodes*. We observe that even a few V-shaped or distance-k V-shaped nodes in the partitioning solution may significantly increase path hopcounts across the cutline. This suggests improving performance by eliminating such nodes.
2) We propose a new algorithm to eliminate, or at least reduce, V-shaped nodes. Instead of modifying the input of MLPart, we modify the MLPart algorithm itself by changing the gain function. We use a "look-ahead" algorithm reminiscent of CLIP [11] to eliminate distance-k V-shaped nodes. We also propose to reweight the nets whose fanout nodes are "X-nodes" to further reduce the hopcount.
3) Our method is easily implementable within standard FM with little cutsize and runtime penalty. We focus on the flat bipartitioning engine context, but our result can be applied within multilevel or any other framework that invokes standard FM. Our approach also extends easily to multiway implementations. Our experimental results show that this method can achieve an average of 39% hopcount reduction on industry testcases, with negligible implementation effort and negligible impact on cutsize and runtime.

[1]That is to say, any cut net will correspond to an interblock wire, or a wire that has some expected length that depends on the size of the given partitioning instance.

Manuscript received May 28, 2003; revised September 6, 2003. This research was supported in part by the MARCO Gigascale Silicon Research Center. This paper was recommended by Guest Editor C. J. Alpert.

A. B. Kahng is with the Department of Computer Science and Engineering, and the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

X. Xu is with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: xuxu@cs.ucsd.edu).

Digital Object Identifier 10.1109/TCAD.2004.825847

0278-0070/04$20.00 © 2004 IEEE

4) We incorporate the new partitioner into the CapoT placer [19] and evaluate circuit delay using a commercial static timing analyzer, Cadence Pearl v5.1 [21]. The experimental results show that the delay is significantly reduced, with very acceptable impacts on wirelength or runtime.

II. Notation and Problem Formulation

Below, we use the following notation.

- $H(V, E)$ denotes the circuit hypergraph.
- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes representing components (e.g., cells) of the circuit.
- $E = \{e_1, e_2, \ldots, e_m\}$ is the set of signal nets, where each net is a subset of nodes that are electrically connected by a signal.
- V_c is the subset of combinational nodes, and V_s is the subset of sequential nodes (or *FF nodes*); $V_c \bigcup V_s = V$ and $V_c \bigcap V_s = \phi$.
- A *bipartition* $(V_0|V_1)$ of $H(V, E)$ divides V into two disjoint subsets V_0 and V_1, such that $V = V_0 \bigcup V_1$; the two subsets are also called *Part 0* and *Part 1*.
- For a net $e = \{v_1, \ldots, v_l\}$, where v_1 is the fanout node whose output signal is the input signal to v_j $(2 \leq j \leq l)$, we say that v_1 is the *input* of v_j $(2 \leq j \leq l)$, and that each v_j $(2 \leq j \leq l)$ is an *output* of v_1.
- If node v_i is an input of node v_j, then we say that there is a *directed edge* from v_i to v_j.
- *PI* denotes the set of primary inputs, and *PO* denotes the set of primary outputs. For purposes of path timing analysis we treat the nodes of *PI*, *PO*, and *FF* as the end points of timing paths, i.e., the *circuit delay* is the longest combinational path delay from any FF or PI output to any FF or PO input. We generically refer to timing paths as *FF-FF* paths.
- $P = (v_{p_1}, \ldots, v_{p_l})$ is a *directed path* from v_{p_1} to v_{p_l}, if there exists a directed edge from $v_{p_{j-1}}$ to v_{p_j}, $(2 \leq j \leq l)$. We say that the *length* of P is $l - 1$.
- Let $P = (v_{p_1}, \ldots, v_{p_l})$ be a directed path from v_{p_1} to v_{p_l}. If $v_{p_j} \in V_c$, $(2 \leq j \leq l-1)$, P is a *combinational directed path*.
- Let $P = (v_{p_1}, \ldots, v_{p_l})$ be a directed path from v_{p_1} to v_{p_l}. If $v_{p_1}, v_{p_l} \in V_s$ and $v_{p_j} \in V_c$, $(2 \leq j \leq l-1)$, P is a *FF-FF path*.
- A combinational node $v_i \in V_c$ is a *distance-k V-shaped node*, or *$V_{(k)}$-node*, if it satisfies: 1) $\exists v_j$, v_t, such that there is a directed edge from v_j to v_i and a combinational directed path from v_i to v_t whose length is k and 2) v_j and v_t are both in the other partition with respect to v_i. For the special case of $k = 1$, v_i is called a *V-shaped node*.
- A combinational node $v_i \in V_c$ is an *X-shaped node*, or *X-node*, if it satisfies: 1) $\exists v_{j0}, v_{t0} \in V_0$ and $v_{j1}, v_{t1} \in V_1$ and 2) there are directed edges from v_{j0} and v_{j1} to v_i, and from v_i to v_{t0} and v_{t1}.
- $A =$ the total area of all the nodes in V. $A_0(\text{resp.}, A_1) =$ the total area of all the nodes in V_0 $(\text{resp.}, V_1)$.
- If $\exists v_i, v_j \in e$, such that $v_i \in V_0$ and $v_j \in V_1$, then e is a *cut net* of the bipartition $(V_0|V_1)$.
- The *cutset* of a bipartition is $E((V_0|V_1)) = \{e \in E | e \text{ is a cut net of } (V_0|V_1)\}$. The *cutsize* of the bipartition is $|E((V_0|V_1))|$.
- A directed edge from v_i to v_j is a *hop*, if v_i is the input to v_j in a cut net.
- $h(P)$ denotes the *hopcount* of an FF-FF path P, i.e., the number of hops in P.
- $h =$ the maximum value of $h(P)$ over all FF-FF paths in $H(V, E)$.
- A *critical path* is an FF-FF path whose hopcount is equal to h.
- Suppose a bipartition $(V_0|V_1)$ has a *cut* on e, and let v_i be the fanout node whose output signal is the input to the rest of the nodes in e. If $v_i \in V_0$, the *cut direction* is indicated as i; if $v_i \in V_1$, the cut direction is indicated as o.

Performance-Driven Bipartition Problem (PDBP)

Given:

Hypergraph $H = (V, E)$

Area balance tolerance $s(0 < s < 1)$, a given parameter that constrains partition areas

$\alpha(0 \leq \alpha \leq 1)$, a given parameter which captures the desired tradeoff between cutsize and path delay in the objective function

Find:

A bipartition $(V_0|V_1)$ which satisfies

$$(1 - s) \cdot \frac{A}{2} \leq A_0 \leq (1 + s) \cdot \frac{A}{2}$$

and minimizes

$$\alpha \left| E\big((V_0|V_1)\big) \right| + (1 - \alpha) h\big((V_0|V_1)\big).$$

III. Previous Performance Driven Methods

Most previous performance-driven partitioning approaches alter the netlist using logic replication, retiming or buffer insertion to meet delay constraints while minimizing the cutsize [7]–[10], [16]. For example, Cong *et al.* [9] propose a global clustering-based partitioning algorithm. The basic idea is as follows.

- Construct a clustered circuit with the minimum clock period, and perform retiming and node duplication as possible.
- Perform cutsize-driven clustering on the clusters formed in the previous step.
- Perform simultaneous cutsize and delay refinement during cluster decomposition.

The method reduces delay by 16% while increasing cutsize by 17%—*with retiming*—compared with hMetis [9]. However, such methods can require substantial gate replication, which potentially increases die area. Some of these methods [7] tend to produce noticeably worse cutsize compared to multilevel FM partitioning.

Other approaches [2], [13] have been proposed which do not change the netlist. Typically, a multilevel FM partitioner such as hMetis [15] is used with some modified (weighted) input. In the taxonomy of [2], these approaches can be divided into

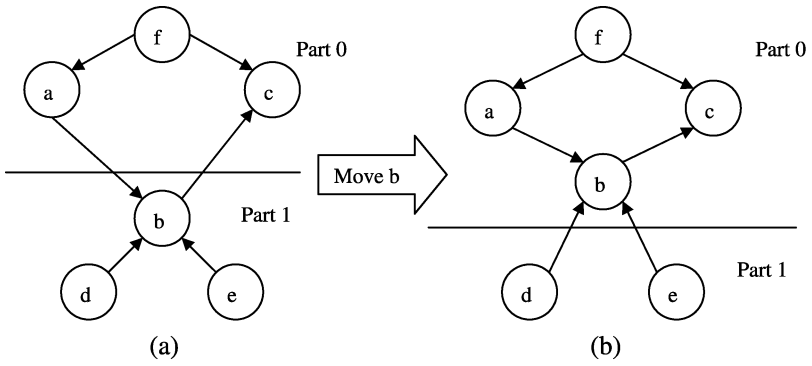


Fig. 1. Example of V-shaped node b.

net-based and *path-based* categories. Net-based partitioning approaches ([18], according to [2]) define a criticality value for each net after timing analysis, while path-based approaches consider the criticality of paths instead of single nets [2]. All of these methods require timing analysis in order to find critical nets or paths, and then reweight the critical nets or paths so as to reduce the chances of cuts occurring. According to Ababei *et al.* [2], their bipartitioning algorithm can reduce delay by 14% at the expense of an increase of 10% in cutsize and 139% in runtime, compared with hMetis [2].

We observe that timing analysis can take a long time and that it is necessarily based on an inaccurate delay model. Delay models such as that of Cong *et al.* [9] (node delay $= 1$, intrablock delay $= 0$, and interblock delay $= 5$) may identify "critical paths" that incorrectly drive the timing analysis and, hence, the partitioner. Since we may not have enough information at the partitioning stage to make accurate delay estimates, for some testcases, these algorithms can produce partitioning solutions with worse delay than solutions found by generic multilevel FM partitioning.[2]

Recently, some algorithms have been proposed which attempt to incorporate timing analysis results somewhat more directly into the FM partitioner [1]. For example, Ababei *et al.* [1] propose to perform timing analysis first in order to assign a "criticality" value to each edge. Then, the gain function of the FM partitioner is changed such that edges with higher criticality will not be cut. Since criticality is a global variable, this means that to obtain a reasonable value of criticality, global timing analysis is needed. Moreover, since the change of one node in the partitioning solution may affect the criticality values of many other nodes, the complexity and convergence of the approach become difficult, as witnessed by the level of improvements reported.

IV. Solving PDBP via Elimination of $V_{(k)}$-Nodes

In a bipartition $(V_0|V_1)$, if all nets in the cutset have the same cut direction, then we call $(V_0|V_1)$ a *unidirectional bipartition*. A key intuition stems from the fact that in a unidirectional bipartition, the hopcount of any FF-FF path is at most one. So, unidirectional bipartitioning is in some sense an "idealized goal" for performance-driven partitioning, and can be sought by, e.g., flow-based methods [17] and KAFM algorithm [5], [6]. Unfortunately, a unidirectional bipartition tends to have much higher cutsize than multilevel FM solutions [6], and for some testcases no purely unidirectional solution exists. Therefore, we propose to relax the unidirectional condition to "locally unidirectional." We call a bipartition $(V_0|V_1)$ without any $V_{(k)}$-nodes (defined in Section II above) as a *distance-k unidirectional bipartition*. Our intuition is that a tradeoff between cutsize and delay can be achieved by elimination or reduction of $V_{(k)}$-nodes in the partitioning solution.

[2]Of course, such models can be very relevant for, e.g., multifield programmable gate array partitioning applications that were prominently studied during the early 1990s.

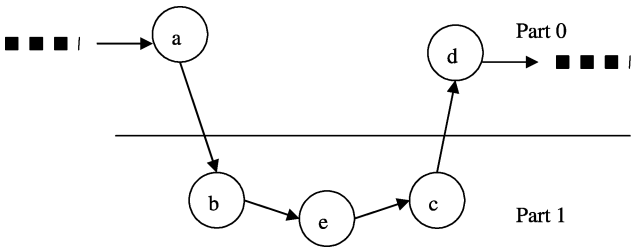


Fig. 2. Example of $V_{(3)}$-node.

Procedure to Calculate $r_j(v_i)$

```
1.  If ((v_i) ∈ V_c) and (In(v_i) ≠ φ))
2.  {
3.   Perform BFS from v_i for at most k levels. At each level j,
     delete all sequential nodes and store the remaining nodes
     before going to the next level.
4.   For (j = 1; j ≤ k; j++)
5.   {
6.    If ((∃ one node ∈ V_1 at level j) and (In(v_i) ⋂ V_1 ≠ φ))
7.        r'_j(v_i) = 1
8.     Else r'_j(v_i) = 0
9.    If ((∃ one node ∈ V_0 at level j) and (In(v_i) ⋂ V_0 ≠ φ))
10.       r''_j(v_i) = 1
11.    Else r''_j(v_i) = 0
12.  r_j(v_i) = r'_j(v_i) − r''_j(v_i)
13.  }
14.}
```

Fig. 3. Procedure to calculate $r_j(v_i)$.

A. V-Shaped Node Elimination

For any node v_i, let $I(v_i)$ be the set of nets to which v_i is connected that lie entirely in the current partition of v_i, and let $O(v_i)$ be the set of nets that belong to the cutset and for which v_i is the only incident node in the partition of v_i. The traditional gain function in FM partitioning is $g(v_i) = |O(v_i)| - |I(v_i)|$ for all nodes v_i. FM partitioning [12] starts with a random initial

Distance-k Unidirectional Bipartitioning Algorithm

1. Calculate initial gains for all nodes and store the gains into two gain buckets B_0 and B_1.
2. Select the node v_i with the maximum gain.
3. Reset the gains of all nodes to zero.
4. Move v_i and update the gains of v_i and its neighbors.
5. **While** ($\exists$ one node that has not been moved)
6. {
7. Select one node v_j with maximum updated gain whose movement satisfies the area balance constraint.
8. Move v_j and update the gains of v_j and its neighbors.
9. Lock v_j and record its gain.
10. }
11. Find the point in the move sequence at which the sum of gains is maximum. Undo all the moves after this point.

Fig. 4. Bipartitioning algorithm.

partition and iteratively checks the node with maximum gain to see whether moving it to the other part will violate the area balance constraint. If not, the node is moved to the other part, otherwise, the node with maximum gain in the other subset will be moved. Every node is locked after moving, and the process continues until all nodes are locked. Then, all prefix sums $S_t = \sum_{j=1}^{t} g(v_j)$ are calculated, and q is chosen such that S_q is maximum (all node moves after the q^{th} are undone). This process is called a *pass*, and FM partitioning repeats passes until $S_q \leq 0$.

In general, the partitioning solutions returned by a multilevel FM partitioner, such as MLPart [4], have good cutsize. However, for some testcases, MLPart tends to produce solutions with high h values. We have analyzed critical paths and consistently found that:

1) there are a few V-shaped nodes in the partitioning solutions;
2) every V-shaped node is included in many critical paths;
3) almost every critical path contains one or more V-shaped node;
4) MLPart cannot eliminate these V-shaped nodes due to the traditional gain function.

Based on these observations, we propose to improve timing by local biasing of FM to eliminate some (not necessarily all) V-shaped nodes. For example, in Fig. 1(a), node b is a V-shaped node. Suppose that the area balance tolerance is 0.35. MLPart cannot move node b to Part 0 since there are directed edges from nodes d and e to b. The traditional gain values of nodes a–f are 0, 0, 0, -1, -1, -2. Since the smallest cutsize is already achieved and no improvement is available, the FM partitioner will stop here. Of the directed paths passing through b, $\ldots a \rightarrow b \rightarrow c \ldots$ will have two cut nets; $\ldots d \rightarrow b \rightarrow c \ldots$ and $\ldots e \rightarrow b \rightarrow c \ldots$ will each have one cut net. However, if we move node b to Part 0, as shown in Fig. 1(b), although the cutsize remains the same, the two cut nets on the path $\ldots a \rightarrow b \rightarrow c \ldots$ are saved, while the numbers of cut nets on the other two paths remain at one. We see that unlike timing-analysis based algorithms, which may increase the hopcounts of near-critical paths when the hopcounts of critical paths are reduced, elimination of V-shaped nodes can improve timing without any negative effect.

We believe that this effect is increasingly important in recent industry testcases: partitioning solutions have much worse timing if they do not consider V-shaped nodes. However, this effect is not apparent for testcases in which most gates have only two inputs. For example, in Fig. 1(a), if node e is removed,

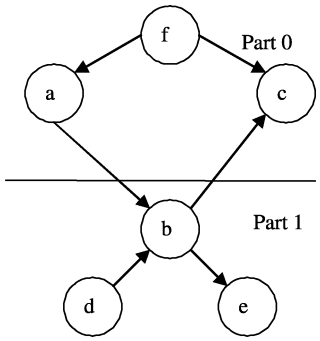


Fig. 5. Example of X-shaped node b.

TABLE I
BASIC PROPERTIES OF TESTCASES

Testcase	#Nets	#Nodes	#FF	#PI	#PO
industry1	16377	15768	729	599	1
industry2	21947	21219	3630	1238	256
industry3	32699	28546	16242	815	628
industry4	21200	21397	2713	164	175
industry5	10217	9794	2426	208	66
industry6	7895	6939	561	294	72

the gain value of node b will be 1, and the FM partitioner will move node b to Part 0. For such testcases, there will be very few V-shaped nodes in the MLPart partitioning solution. Our method may, therefore, be more suited to the "true" underlying netlist topology after synthesis, and in fact is not effective if the netlist has been reduced to some sort of generic 2-input gate variant.[3]

B. Elimination of Generalized V-Shaped Nodes

For some testcases, eliminating V-shaped nodes is not sufficient since there are many $V_{(2)}$-nodes and $V_{(3)}$-nodes left in the partition solutions after elimination of V-shaped nodes. Moving these nodes can make the solutions better. Ideally, we hope that no two cut nets with different directions (one i and one o cut net) are located too close to each other in a path.[4] Therefore, we want to eliminate $V_{(k)}$-nodes, with k as large as possible. However, the number of $V_{(k)}$-nodes will increase dramatically with k, which means that we likely need to change the pure

[3]Note that ISCAS sequential benchmarks have been broken down into 2-input gates, and thus are not an interesting context for us.

[4]The intuition is as follows. For a FF-FF path P of 50 nodes, the max possible hopcount is 49. If we require that the distance between any two opposite-direction cuts is no smaller than 5, the max possible hopcount is 9. By making k large enough, we can force the decrease of $h((V_0, V_1)) = 1$ to its minimum value.

TABLE II
BIASING AGAINST V-SHAPED NODES VERSUS MLPART: DETAILED RESULTS OF TEN INDEPENDENT SINGLE-START TRIALS FOR THE TESTCASE INDUSTRY1. THE PARAMETERS USED IN OUR ALGORITHM ARE $\delta(0) = 1$ AND $\delta(1) = 10$; AREA BALANCE TOLERANCE IS 10%. $change = (New_cutsize - MLPart_cutsize)/(MLPart_cutsize)$; $improve = (MLPart_delay - New_delay)/(MLPart_delay)$

Trial	MLPart [4]				MLPart+V-nodes Removal					
	cutsize	h	delay	CPU(s)	cutsize	change	h	delay	improve	CPU(s)
1	803	5	330.4	11.97	832	3.6%	3	262.1	20.7%	12.14
2	821	6	378.5	11.88	853	3.9%	4	277.6	26.7%	12.45
3	815	5	330.4	12.03	844	3.5%	3	262.1	20.7%	12.38
4	817	6	378.5	11.65	848	3.6%	4	277.6	26.7%	12.67
5	839	5	330.4	11.56	844	0.6%	3	262.1	20.7%	13.01
6	822	5	330.4	11.63	852	3.6%	3	262.1	20.7%	12.91
7	823	5	330.4	11.89	852	3.5%	3	262.1	20.7%	12.32
8	841	6	378.5	11.64	863	2.6%	4	277.6	26.7%	12.31
9	798	5	330.4	11.71	824	3.3%	3	262.1	20.7%	12.42
10	828	5	330.4	11.92	847	2.3%	3	262.1	20.7%	12.57

mincut-driven solution more substantially with large k. Another problem associated with large k is that movement of one node may affect many other nodes, again leading to increased cutsize and complexity.[5] Currently, we do not believe that it is practical to be concerned with $k \geq 4$.

To eliminate $V_{(k)}$-nodes, we effectively need to move more than one node. For example, in Fig. 2, we need to move nodes b, c, e in order to eliminate $V_{(3)}$-node b. If just node b is moved, e will become a new $V_{(2)}$-node. For any $V_{(k)}$-node v_i, we denote the set of nodes whose movement is required to eliminate v_i as $MS(v_i)$ and we require $v_i \in MS(v_i)$. In Fig. 2, $MS(b) = \{b, c, e\}$.[6] Therefore, after moving one node v_i, we need to use a "look ahead" algorithm to, in effect, move the rest of the nodes in $MS(v_i)$. we achieve this by means similar to the CLIP algorithm of Dutt *et al.* [11]. That is, we reset the gains of all nodes to zero after initial gain calculation such that the gain of a node after update only reflects its goodness for moving with regard to the nodes currently being moved. For example, in Fig. 2, after moving the node b and gain update, the node e has the largest gain.

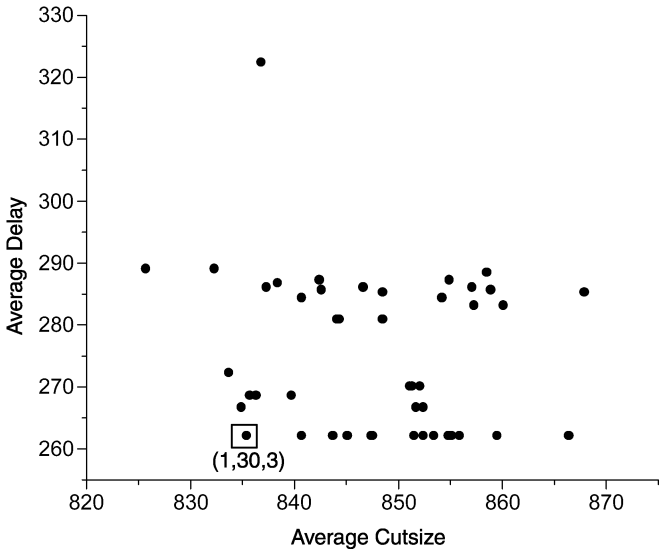


Fig. 6. Scatter plot for the testcase industry1. Each round point represents the (cutsize, delay) pair for one combination of $\delta(0)$, $\delta(1)$, and $\delta(2)$, whose values are chosen from {1, 3, 10, 30}.

C. New Gain Function Calculation

To achieve what we call "distance-k unidirectional bias", we change the gain function to: $\text{Gain}(v_i) = \delta(0)g(v_i) + \sum_{j=1}^{k} \delta(j) r_j(v_i)$ for each node v_i

Here, $g(v_i)$ is the traditional net-cut based gain function. The user-defined coefficients $\delta(j) \geq 0$ weight the attention paid to different V-shapes (if only $\delta(0)$ is positive, then we have the original FM algorithm), and $r_j(v_i)$ is the reduction in the number of $V_{(j)}$-nodes after moving v_i. For any node $v_i \in V_c$, we denote the set of inputs of v_i as $In(v_i)$. To simplify the description, we assume that $v_i \in V_0$. The procedure to calculate $r_j(v_i)$ is shown in Fig. 3.

In Steps 1–3 of the procedure, if v_i is a combinational node and its input set is not empty, we find all the nodes within distance k of v_i using BFS. All sequential nodes and their descendants will be removed from the BFS tree. Then, for every level j from 1 to k, we check whether v_i is a $V_{(j)}$ node in Steps 6–8. If so, $r'_j(v_i) = 1$. In Steps 9–11, we check whether v_i will be a $V_{(j)}$ node if it is moved to the other part. The value of $r_j(v_i)$ can be 1, 0, -1, which represents the reduction of the number of $V_{(j)}$ nodes in the hypergraph due to the move of v_i.

To analyze the time complexity of the procedure, assume that the maximum fanin is MI and the maximum fanout is MO. For every node, checking the inputs will take at most $O(MI)$ time and BFS will take at most $O(MO^k)$ time. Therefore, the total time needed for calculating $r_j(v_i)$ is $O(MO^k + MI)$.

Distance-k Unidirectional Bipartitioning: Summary and Time Complexity: Our algorithm to achieve distance-k unidirectional bias in the bipartitioning (reminiscent of CLIP [11] in how it induces movement of clusters across the cutline) is summarized in Fig. 4. Time complexity may be analyzed as follows. We use the same gain bucket list structure as proposed in [12]. The maximum possible gain is $\text{Gain}_{\max} = \delta(0)g_{\max} + \sum_{j=1}^{k} \delta(j)$, where $g_{\max}$ is the maximum possible traditional gain. The time for calculating initial gain is $O((MO^k + MI)n)$, where n is the number of nodes in the hypergraph. $O(\text{Gain}_{\max})$ time is needed to reset the gains of all nodes to zero, since we only need to remove all linked lists from buckets and concatenate them to the bucket

[5]In fact, the experimental results for $k = 3$ and $k = 4$ show that the runtime and cutsize become worse while the delay is not improved compared with results of $k = 2$.

[6]We require $v_i \in MS(v_i)$ in our approach. Other approaches such as moving nodes a or d are not considered since nodes a and d may have neighbors and we wish to avoid the complexity of checking all possible configurations.

TABLE III
RESULTS OF MLPART [4] AND REWEIGHTING [2]: AVERAGE RESULTS OF TEN RANDOM STARTS. $change = (New_cutsize - MLPart_cutsize)/(MLPart_cutsize)$; $improve = (MLPart_delay - New_delay)/(MLPart_delay)$

Testcase	MLPart [4]				Reweighting [2]					
	cutsize	h	delay	CPU(s)	cutsize	change	h	delay	improve	CPU(s)
industry1	820.7	5.3	352.8	11.79	851.2	3.7%	4.2	326.4	7.5%	110.5
industry2	169.9	3.5	220.7	13.45	191.4	12.7%	2.7	214.6	2.7%	102.3
industry3	141.3	3	291.6	16.67	152.3	7.8%	2.6	288.4	1.1 %	140.7
industry4	408.7	5.3	302.6	12.43	466.8	14.2%	4.7	288.1	4.8%	112.6
industry5	136.8	4.6	287.5	7.48	163.4	19.4%	4.4	280.3	2.5%	89.7
industry6	338.6	5.1	294.8	9.32	368.2	8.7%	4.9	281.4	4.5%	98.6

TABLE IV
RESULTS OF BIASING AGAINST $V_{(k)}$-NODES (UP TO $k = 2$) AND BIASING AGAINST $V_{(2)}$-NODES PLUS X-NODES REWEIGHTING: AVERAGE RESULTS OF TEN RANDOM STARTS. THE PARAMETERS USED IN OUR METHOD ARE $\delta(0) = 1, \delta(1) = 30, \delta(2) = 3$, AND $\beta = 10$. $change = (New_cutsize - MLPart_cutsize)/(MLPart_cutsize)$; $improve = (MLPart_delay - New_delay)/(MLPart_delay)$

Testcase	$V_{(2)}$-nodes Removal						$V_{(2)}$-nodes Removal+X-nodes Reweighting					
	cutsize	change	h	delay	improve	CPU(s)	cutsize	change	h	delay	improve	CPU(s)
industry1	847.5	3.2%	3	262.1	34.6%	13.16	851.3	3.7%	2.9	242.4	45.5%	17.54
industry2	183.2	7.8%	2	202.5	8.2%	15.67	184.7	8.7%	2	202.5	8.2%	16.03
industry3	149.2	5.6%	2	275.6	5.5%	18.92	148.6	5.1%	1.9	270.8	7.1%	21.47
industry4	416.7	2.0%	3.4	243.5	24.3%	14.79	423.4	3.6%	3.1	221.2	26.9%	18.81
industry5	142.3	4.0%	3.6	236.8	17.6%	8.91	148.9	8.8%	3.4	228.3	20.6%	12.45
industry6	347.7	2.7%	4.1	268.4	8.9%	11.24	362.1	6.9%	3.9	263.7	10.5%	16.63

TABLE V
TIMING-DRIVEN PLACEMENT RESULTS. THE MODIFIED CAPOT USES THE NEW PARTITIONER WITH $V_{(2)}$-NODES ELIMINATION AND X-NODES REWEIGHTING. THE PARAMETERS USED IN THE PARTITIONER ARE $\delta(0) = 1, \delta(1) = 30, \delta(2) = 3$, AND $\beta = 10$

Testcase	Original CapoT[14]			Modified CapoT		
	HPWL(μm)	Slack(ns)	CPU(s)	HPWL(μm)	Slack(ns)	CPU(s)
industry1	5.30e6	0.63	67.2	5.37e6	0.76	79.7
industry2	3.22e6	-0.47	109.8	3.18e6	-0.11	124.2
industry3	7.56e6	1.01	142.6	7.52e6	1.39	171.4
industry4	2.60e6	0.38	87.8	2.60e6	0.41	96.1
industry5	4.46e6	0.91	64.4	4.50e6	1.12	76.2
industry6	1.04e6	-0.74	57.3	1.03e6	-0.58	65.9

of zero gain. Moving gains also takes $O(\text{Gain}_{\max})$ time. Since moving one node only affects the gains of the nodes within distance k, we need to update at most $O(MO^k + MI^k)$ nodes in every iteration. Because every node can be moved at most once, the total time should be $O((MO^k + MI^k + \text{Gain}_{\max})n)$. Therefore, our algorithm takes linear time per pass, just as in the original FM algorithm. The negligible impact on runtime is confirmed in the next section.

D. X-Nodes Reweighting

In our analysis of critical paths, we notice that there are a few X-nodes, as shown in Fig. 5. These are nets that "straddle a cutline" and which should be penalized. Since moving node b from Part 1 to Part 0 will increase the hopcount of the path $d \rightarrow b \rightarrow e$ by two, the reduction of hopcount can not be obtained by simply moving the node b. One node v_i can be identified as X-node if both Part 0 and Part 1 have at least one input and one output of v_i. In order to eliminate X-nodes, we propose to increase the weights of the nets whose fanout nodes are X-nodes, such as the net $\{b, c, e\}$ in Fig. 5 in order to constrain the cutsize driven partitioner not to cut these nets.[7] Performing X-nodes reweighting and V-nodes elimination simultaneously is difficult since both of these operations will change the gain structures for the netlist.[8] Thus, we perform X-nodes reweighting after V-nodes elimination at the expense of the increase of runtime. Initially, the weights of all nets are set as 1. We reset the weight of each net whose fanout node is an X-node to a given constant β. The new gain function is: $\text{Gain}(v_i) = g(v_i) + \beta - 1$ for each node v_i if v_i belongs to one net whose fanout node is an X-node and moving v_i can save the net from being cut; $\text{Gain}(v_i) = g(v_i)$ otherwise. Here, $g(v_i)$ is the traditional gain function of v_i. We then use the algorithm specified in Fig. 4 to obtain the final bipartitioning solution.

[7]For example, if we reweight one net e as ten, the cutsize will increase by ten instead of by one if e is a cut net.

V. EXPERIMENTAL RESULTS

The MLPart code of [4] was downloaded from the MARCO GSRC Bookshelf [20] and modified. The code is currently compiled and run on Solaris and Linux platforms. Total code modifications amounted to less than 2000 lines.

We tested our algorithm on four industry testcases given to us in LEF/DEF format. The testcase parameters are summarized in Table I. All tests were run on code compiled with the GNU gcc2.95.2 compiler running on a 600-MHz Intel Pentium-III Xeon processor under the RedHat7.3 Linux operating system. We use the model in [2] to calculate the delay.

[8]In fact, our attempts to do so did not yield strong results.

Table II shows the results of multiple single-start runs on the testcase "industry1" when run with $\delta(0) = 1$, $\delta(1) = 10$. The results show that around 30% improvement on hopcount and 23% improvement on delay is achieved while only slightly increasing cutsize and runtime.[9]

We also tested our algorithm with different values of $\delta(0)$, $\delta(1)$, and $\delta(2)$ for the testcase "industry1" in order to find the best tuning of parameter values. In each test, we choose a different combination of values from the set $\{1, 3, 10, 30\}$ for each of the three parameters, and run the code with 10 independent random starts. The (cutsize, delay) pairs across all 64 combinations averaged over ten starts for each combination are given as a scatter plot in Fig. 6. The best tradeoff point is achieved with $\delta(0) = 1$, $\delta(1) = 30$, and $\delta(2) = 3$. Empirically, we believe that good results are consistently achieved with $\delta(0) = 1$, $\delta(1) = 30$, and $\delta(2) = 3$.

Table III gives the average results of MLPart [4] and reweighting for all the six testcases with ten random starts, comparing directly against the results of $V_{(2)}$-nodes removal and $V_{(2)}$-nodes removal plus X-nodes reweighting in Table IV.[10] We set $\delta(0) = 1$, $\delta(1) = 30$, $\delta(2) = 3$, and $\beta = 10$. The results show that our algorithm is very efficient in reducing hopcount as well as delay. Across all testcases, the increase of cutsize (average of 5.1% after X-nodes reweighting) and runtime (average of 30.9%) is acceptable.

Finally, to address our original motivating application, we have studied the impact of the new partitioner within the framework of top-down, partitioning-based, timing driven placement. We have incorporated the new partitioner into CapoT [19], a timing driven placer used in [14]. Table V compares the results of modified CapoT by using the new partitioner with the original CapoT. Circuit delay is evaluated by a commercial static timing analyzer, Cadence Pearl v5.1 [21]. The experimental results show that worst-case timing slack is increased with the new partitioner, while the increase of wirelength (average of 0.1%) and runtime (average of 15.9%) is quite moderate.

VI. Conclusion

In this paper, we have proposed a simple yet efficient timing-driven partitioning algorithm which does not rely on any global timing analysis. Since only local information is used in the algorithm, we achieve an effective return of solution quality versus runtime. By changing the gain function in the FM partitioner, we bias toward movement of some $V_{(k)}$-nodes in the FM partitioning solution across the cutline. We have observed that these "bad nodes," that is, $V_{(k)}$-nodes, contribute significantly to the delay of the whole circuit. Thus, our biasing approach improves timing by eliminating or minimizing such nodes. We also propose to reweight the nets whose fanout nodes are X nodes to further reduce the hopcount and path delay. Experimental results show that our method significantly reduces path delay while keeping the cutsize and runtime almost the same as MLPart. To verify the effectiveness of our new partitioner, it is incorporated into a placer and results are evaluated by a commercial static timing analyzer. We observe that the circuit delay is reduced while the wirelength remains almost the same and the increase of runtime is moderate.

[9]When using the simpler delay model of [9], we obtain an average of 20% improvement in delay over all testcases.

[10]Programs described in [9] and [2] were not available for comparison. The reweighting code is obtained by modifying MLPart [20] according to the algorithm proposed in [2].

References

[1] C. Ababei and K. Bazargan, "Statistical timing driven partitioning for VLSI circuits," in *Proc. Design Automation Test Eur.*, 2002, p. 1109.

[2] C. Ababei, S. Navaratnasothie, K. Bazargan, and G. Karypis, "Multi-objective circuit partitioning for cutsize and path-based delay minimization," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 2002, pp. 181–185.

[3] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: Method for heuristic development, experimentation and reporting," in *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 349–354.

[4] ——, "Improved algorithms for hypergraph bipartition," in *Proc. Asia South Pacific Design Automation Conf.*, 2000, pp. 661–666.

[5] J. Cong, Z. Li, and R. Bagrodia, "Acyclic multi-way partitioning of Boolean networks," in *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 670–675.

[6] J. Cong, W. Labio, and N. Shivakumar, "Multi-way VLSI circuit partitioning based on dual netlist representations," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 1994, pp. 56–62.

[7] J. Cong, H. Li, and C. Wu, "Simultaneous circuit partitioning/clustering with retiming," in *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 460–465.

[8] J. Cong, S. Lim, and C. Wu, "Performance driven multi-level and multiway partitioning with retiming," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 274–279.

[9] J. Cong and C. Wu, "Global clustering-based performance driven circuit partitioning," in *Proc. ACM/IEEE Int. Symp. Physical Design*, 2002, pp. 149–154.

[10] W. E. Donath, R. J. Norman, B. K. Agrawal, S. E. Bello, S. Y. Han, J. M. Kurtzberg, P. Lowy, and R. I. McMillan, "Timing driven placement using complete path delays," in *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 84–89.

[11] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 194–200.

[12] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Automation Conf.*, 1982, pp. 175–181.

[13] Y. C. Ju and R. A. Saleh, "Incremental techniques for the identification of statically sensitizable critical paths," in *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 541–546.

[14] A. B. Kahng, S. Mantik, and I. L. Markov, "Min-max placements for large-scale timing optimization," in *Proc. ACM/IEEE Int. Symp. Physical Design*, 2002, pp. 143–148.

[15] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526–529.

[16] L. Liu, M. Shih, N. Chou, C.-K. Cheng, and W. Ku, "Performance-driven partitioning using a replication graph approach," in *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 206–210.

[17] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 1998, pp. 497–504.

[18] S. L. Ou and M. Pedram, "Timing-driven partitioning using iterative quadratic programming," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 472–476.

[19] Capo: A large-scale fixed-die placer Available: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/Capo/ [Online]

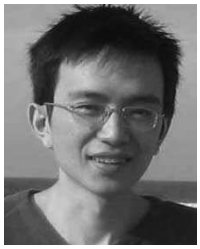
[20] MLPart: High-performance multilevel hypergraph bipartitioning code Available: http://nexus6.cs.ucla.edu/GSRC/bookshelf/Slots/Partitioning/MLPart/ [Online]

[21] Cadence Pearl Available: http://www.ece.uci.edu/eceware/cadence_docs/pearluser/ [Online]

[22] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics is from Harvard College and the M.S. and Ph.D. degrees in computer science are from University of California (UC), San Diego.

From 1989 to 2000, he was a member of the UC, Los Angeles computer science faculty. He is a Professor of CSE and ECE at UC, San Diego. He has published over 200 papers in the VLSI CAD literature, receiving three Best Paper awards and an NSF Young Investigator award. His research is mainly in physical design and performance analysis of VLSI, as well as the VLSI design-manufacturing interface. Other research interests include combinatorial and graph algorithms, and large-scale heuristic global optimization. Since 1997, he has defined the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and since 2001 has chaired the U.S. and international working groups for Design technology for the ITRS. He has been active in the MARCO Gigascale Silicon Research Center since its inception. He was also the founding General Chair of the ACM/IEEE International Symposium on Physical Design, and cofounded the ACM Workshop on System-Level Interconnect Planning.



Xu Xu was born in Maanshan, China, in 1975. He received the B.S. degree from the University of Science and Technology of China, Hefei, in 1998. He is currently pursuing the Ph.D. degree in computer science and engineering from the University of California at San Diego, La Jolla.

He was with Ammocore Technology, Inc., Santa Clara, CA, in 2002. His research includes VLSI timing optimization, application of VLSI algorithms on DNA arrays, and mask-manufacturing cost minimization.