Timing-Driven Bipartitioning with Replication Using Iterative Quadratic Programming

Shihliang Ou and Massoud Pedram
Department of Electrical Engineering - Systems
University of Southern California, Los Angeles, CA 90089
Email: {shihlian, massoud}@zugros.usc.edu

Abstract
We present an algorithm for solving a general min-cut, two-way partitioning problem subject to timing constraints. The problem is formulated as a constrained programming problem and solved in two phases: cut-set minimization and timing satisfaction. A mathematical programming technique based on iterative quadratic programming (TPIQ) is used to find an approximate solution to the constrained problem. When the timing constraints are too strict to have a feasible solution, node replication is used to satisfy the constraints. Experimental results on ISCAS89 benchmark suite show that TPIQ can solve the timing-driven bipartitioning problem with little impact on the chip size.

1. Introduction

Partitioning plays an important role in VLSI physical design. It affects not only the efficiency of the subsequence design processes, but also the overall circuit performance. In the past, most of the partitioning algorithms focused on minimizing the cut size. With the advances in CMOS process technologies, the difference between intra-part delays (internal delay) and inter-part delays (external delay) increases. A partitioning solution which results in cutting the critical paths of a circuit too many times may not meet the timing requirements of that system. Thus, a timing-driven partitioning algorithm for a high performance circuit must reduce the cut size as well as minimize the number of times the critical paths are cut.

The main source of difficulty in timing-driven partitioning is the need for considering all paths of the circuit at the same time. Without this global view, minimizing the delays of some critical paths may result in creating other critical paths. Most of the bipartitioning techniques which are reported in the literature are based on Fiduccia-Mattheyses (FM) algorithm [1]-[5]. FM-based heuristics maintain a local view of the circuit and hence do not appear to be suitable for handling the timing constraints [6]. Other approaches [7]-[11] have been proposed to cope with this problem. The methods in [7]-[9] focus on delay improvement only whereas the computational efficiency of the methods in [10][11] is a concern.

In this paper, we propose a general timing-driven bipartitioning algorithm called TPIQ. The idea used in TPIQ is to transform the Boolean quadratic programming formulation of a timing-driven partitioning problem to an iterative quadratic programming problem in continuous space. By incrementally constraining the variables, the solution will gradually converge to discrete space. TPIQ consists of two phases. The first phase formulates the problem as a capacity-constrained quadratic programming problem which aims at minimizing the cut size without considering the timing constraints. The second phase starts with the result of phase I as the initial solution and identifies nodes which violate the timing constraints. These critical nodes are reassigned in a way similar to phase I, but this time under appropriately enforced timing constraints.

In both phases, a unified technique based on iterative quadratic programming (IQP) is used. IQP simulates the Boolean quadratic programming by iteratively adding extra edges to connect nodes in the circuit to two dummy fixed nodes: one placed at 1, and the other placed at 0. The final solutions of IQP are moved to either 1 or 0 if the assignments meet the timing constraints. Otherwise, the nodes remain floating between 0 and 1. Nodes with non-binary final solution are replicated and assigned to both parts so as to minimize delay. This algorithm was tested on MCNC ISCAS89 benchmark suite ranging from 554 to 23,949 nodes. The result indicates that TPIQ is capable of minimizing cut size while meeting the delay requirements with negligible node replication.

The remainder of the paper is organized as follows: Section 2 describes the problem. Section 3 presents the iterative quadratic programming partitioning technique. The formulation of constraints and the replication method are given in section 4. Section 5 provides our experimental results. Section 6 concludes this paper and describes further research directions.

2. Problem Statement

The timing-driven partitioning problem can be stated as follows. Consider a circuit (combinational or sequential) which is represented by a directed graph $G=(V, E)$. Each node $v_i \in V$ has a weight $\delta(v_i)$ which specifies the intrinsic delay associated with v_i. Each edge $(v_i, v_j) \in E$ has a wiring delay $d(v_i, v_j)$. If (v_i, v_j) is cut, $d(v_i, v_j)=D$ where D is the external delay, otherwise, $d(v_i, v_j)$ is the internal delay inside a part. The problem is to partition the nodes into two groups so as to minimize the cut size while satisfying the user-specified timing and capacity constraints. This problem can be simplified further by replacing the internal delays with a constant d. This simplification is often justified since the location of nodes in a part and the lengths of connections within a part are unknown at this stage. In general, $D >> d$.

The cut-set $\vartheta = \{\vartheta_1, \vartheta_2 \ldots \vartheta_k\}$ is used to define a delay function for the circuit. Let $c(p,\vartheta)$ be the number of cut edges along a path p, and $|p|$ be the number of nodes on the path p. The delay $d_V(v_i)$ of an output node v_i is:

$$d_V(v_i) = \max_p \left\{ \sum_{v_i \in p} \delta(v_i) + Dc(p, \upsilon) + d(|p| - 1 - c(p, \upsilon)) \right\}$$

for all paths p ending at v_i.

The cycle time for the graph G: $\Phi_G = max\ d_V(V)$ is a func-

This research was supported in part by the Semiconductor Research Corporation under contract no. 98-DJ-606.

tion of $c(p,\vartheta)$. For the timing constrained problem, each primary output node or the input of a register is annotated with a required time. A critical path is a path whose output arrival time exceeds the specified required time. To eliminate the critical paths, every $c(p,\vartheta)$ must be within a range so that $d_V(v_i)$ is smaller than or equal to the corresponding required time.

3. Iterative Quadratic Programming

Let $C=[c_{ij}]_{NxN}$ denote the adjacency matrix of a graph G, where c_{ij} is the connectivity weight or the cost between nodes v_i and v_j. An assignment function $\varphi(v_i)$ assigns v_i to either part 0 or part 1:

$$\varphi(v_i) = \begin{cases} 0 & if\ v_i\ is\ assigned\ to\ part\ 0 \\ 1 & if\ v_i\ is\ assigned\ to\ part\ 1 \end{cases}$$

The objective function is defined as:

$$f(V)=\frac{1}{2}\sum_{v_i, v_j \in V} c_{ij}(\varphi(v_i)-\varphi(v_j))^2$$

Let $x_i = \varphi(v_i)$, $X=\varphi(V)=[x_1, x_2, \ldots x_N]^T$. The objective function becomes: $f(X)=\frac{1}{2}\sum_{x_i, x_j \in X} c_{ij}(x_i - x_j)^2$.

The function can be written in a matrix notation: $f(X)=\frac{1}{2}X^T QX$, where $Q=B-C$ is the *Laplacian* matrix and $B=[b_{ij}]_{NxN}$ is the degree matrix with $b_{ii}=\deg(v_i)$, and $b_{ij}=0$ if $i \neq j$.

This is a Boolean quadratic programming problem formulation. To approximate this problem, we relax the Boolean restriction on x_i. Every variable x_i in X can be a real number in [0,1]. The objective is to minimize

$$f(X)=\frac{1}{2}X^T QX\ ,\ 0 \le x_i \le 1 \text{ for all } x_i \in X.$$

The matrix Q which appears in the quadratic term is positive semi-definite hence the function $f(X)$ is convex.

In spite of the removal of the binary restriction for x_i's, a final feasible solution for this problem is one in which every $x_i \in \{0, 1\}$. To encourage x_i to converge to a binary value, an attractive force from boundary 0 or 1 is added. The attractive force comes from two fixed dummy nodes: v_{s0} which is fixed at 0 and v_{s1} which is fixed at 1.

We give some terminology first. An *anchored node* is a node which is connected to exactly one fixed node. A *free node* is a node which does not connect to any fixed node. Let P_0 = {the set of anchored nodes which connect to v_{s0}}, P_1 = {the set of anchored nodes which connect to v_{s1}}, P_f = {the set of free nodes}.

If the solution x_i of one iteration of the quadratic program is "close" to 0, v_i is likely to be assigned to P_0. Therefore, an extra edge is added to connect v_{so} and v_i, and v_i becomes a left-anchored node. A neighborhood region R_0 is used to determine whether v_i is "close" enough to be attracted to v_{s0}. Similarly if x_i falls into a neighborhood region R_1, it becomes a right-anchored node. If no node falls into R_0 or R_1, a node which is closest to 1 or 0 is selected to be anchored. The new configuration is passed as input to the next iteration of quadratic program until all nodes are anchored to either v_{s0} or v_{s1}.

The objective function $f(X)$ for the new configuration is:

$$\frac{1}{2}\sum_{x_i, x_j \in X} c_{ij}(x_i - x_j)^2 + c_{max}\left[\sum_{v_i \in P_o}(x_i - 0) + \sum_{v_i \in P_1}(1 - x_i)\right]$$

$$,\ 0 \le x_i \le 1 \text{ for all } x_i \in X.$$

c_{max} is the weight of the edge connecting an anchored node to its respective fixed node. It must be large enough to ensure that every x_i converges to the boundary values. At the same time, it should not be too large to overshadow the first term. We thus seek a minimum c_{max} such that the second term in the above equation is always equal to or greater than the first term. Each $v_i \in P_1$ contributes $g(x_i, x_j)$ to the objective function:

$$g(x_i, x_j) = \sum_{j=1}^{N} c_{ij}(x_i - x_j)^2 + c_{max}(1 - x_i)\ ,\ c_{ij} \neq 0.$$

If we differentiate $g(x_i, x_j)$ with respect to x_i, and let $g'(x_i, x_j)=0$, then we obtain: $\sum_{j=1}^{N} 2c_{ij}(x_i - x_j) - c_{max} = 0$

When v_i is anchored to P_1 and all the other v_j's ($c_{ij} \neq 0$) are anchored to P_0, the c_{max} for v_i is $2\cdot\Sigma c_{ij}$, $j=1..N$. Hence, in the worst case, the minimum c_{max} for v_i must be equal to or larger than $2\cdot\Sigma c_{ij}$ to ensure v_i will have a final solution of 1. We can derive the same c_{max} for any $v_i \in P_0$. The minimum c_{max} for $f(X)$ is $max\{2\cdot\Sigma c_{ij}, j=1..N, \forall\ v_i \in V\}$.

Let $\overline{V}=V \cup \{v_{s0}, v_{s1}\}=[v_1, v_2 \ldots v_N, v_{s0}, v_{s1}]^T$, $\overline{X}=X \cup \{0, 1\}=[x_1, x_2 \ldots x_N, 0, 1]^T$. $\overline{Q}$ is the new *Laplacian* matrix formed by attaching two columns of all zeros and two rows of all zeros to Q. The assignment function is redefined as:

$$\varphi'_0(v_i) = \begin{cases} 1 & if\ v_i\ is\ anchored\ to\ v_{s0} \\ 0 & otherwise \end{cases},$$

$$\varphi'_1(v_i) = \begin{cases} 1 & if\ v_i\ is\ anchored\ to\ v_{s1} \\ 0 & otherwise \end{cases},$$

and the vector b is given by:

$$b = c_{max}\cdot\left[\varphi'_0(v_1)\ \varphi'_0(v_2)\ \ldots\ \varphi'_0(v_N)\ \sum_{i=1}^{N} -\varphi'_0(v_i)\ \ 0\right]^T$$

$$+ c_{max}\cdot\left[-\varphi'_1(v_1)\ -\varphi'_1(v_2)\ \ldots\ -\varphi'_1(v_N)\ \ 0\ \sum_{i=1}^{N}\varphi'_1(v_i)\right]^T$$

In matrix notation, this results in: $f(X)=\frac{1}{2}\overline{X}^T\overline{Q}\overline{X} + b^T\overline{X}$

In TPIQ, we adopt a new weight function which calculates the c_{ij} weights according to the number of anchored nodes of nets. A net is represented by the clique model. Assuming that the probability distribution for the combinations of assigning free nodes to either side is uniform, then the weight is defined as the reciprocal of the expected value for the net to be cut. Let $n_{k,sum}$ be the sum of cut edges of all assignment combinations for a net n_k which connects n nodes. Note that $n_{k,sum}$ is computed by $\sum_{i=0}^{N_{kf}} i(N_{kf} + N_{ka} - i)$ where N_{kf} is the number of free nodes, and N_{ka} is the number of anchored nodes connected to exactly one fixed node in n_k. The weight c_{ij} of every node pairs in n_k is defined as:

$$\begin{cases} n/\left(\sum_{i=1}^{n} i\cdot(n-i)\right) & \text{if all nodes in } n_k \in P_f \\ N_{kf}/(n_{k,sum}) & \text{if one or more nodes are anchored to exactly one fixed node} \\ 0 & \text{if at least one node is left–anchored and at least one node is right–anchored} \end{cases}$$

A net which connects more nodes has a higher probability of being cut, hence, the weight function gives edges in a larger net smaller weights.

The selection of the first node to be anchored affects the convergence speed of TPIQ. A heuristic is to draw a denser cluster in the graph to one side as early as possible. The way we implement this heuristic is to choose the node with the largest degree as the first anchored node.

4.Timing Constraints

Path delay in TPIQ is determined by block-oriented timing analysis [12]. The timing constraints are:

arrival time for any internal node v_j:

$a_j = max\{a_i + d(v_i,v_j) + \delta(v_i) \mid \forall(v_i,v_j) \in E\}$

The values of arrival time for the primary inputs (PI) or the outputs of registers (PPI, pseudo PI) and the values of required time for primary output (PO) or the input of registers (PPO, pseudo PO) are specified by the user. We have the following inequalities for PI (PPI) and PO (PPO):

$a_j \geq T_{aj}$, $\forall$ $v_j \in$ {set of PIs or PPIs}, where T_{aj} is a prespecified arrival time for v_j,

$r_i \leq T_{ri}$, $\forall$ $v_i \in$ {set of POs or PPOs}, where T_{ri} is a prespecified required time for v_i.

If x_i and x_j are close to one another, they have high tendency to be assigned to the same part and the internal delay d dominates. Otherwise external delay D dominates. Hence, the connection delay $d(v_i,v_j)$ is given by:

$d(v_i,v_j) = D \cdot |x_i - x_j| + d \cdot (1 - |x_i - x_j|)$

Since the nodes are anchored gradually, $d(v_i,v_j)$ approaches either D or d. The arrival time is rewritten as:

$a_j \geq a_i + D \cdot |x_i - x_j| + d \cdot (1 - |x_i - x_j|) + \delta(v_i)$

Let $D = k{\cdot}d$, then we have

$$\begin{cases} (a_j - a_i - \delta(v_i) - d)/((k-1)\cdot d) \geq x_i - x_j \\ -((a_j - a_i - \delta(v_i) - d)/((k-1)\cdot d)) \leq x_i - x_j \end{cases}$$

The delay equations are thus written as linear equations and absolute value signs are removed.

Let $s(v_i)$ denote the size of node v_i. $S_p(v_i) = \Sigma_p\, s(v_i)$ is the sum of node sizes in part $p \in \{P_0, P_1\}$. Then we have:

Lower bound $\leq S_p(v_i) \leq$ Upper bound, $\forall$ $p \in \{P_0, P_1\}$, where the lower and upper bound are user specified.

Furthermore, we have $0 \leq x_i \leq 1$, $\forall$ $x_i \in X$.

The formal formulation for the quadratic programming is:

minimize $\frac{1}{2}\bar{X}^T\bar{Q}\bar{X} + b^T\bar{X}$ *subject to* constraints

Let M_{in} denote the total number of input pins in the graph G, N the number of nodes, N_{reg} the number of registers, N_{input} the number of PIs, and N_{output} the number of POs. For TPIQ phase II, the total number of variables is: $2{\cdot}N + N_{reg}$. The number of constraints is $2{\cdot}M_{in}+2{\cdot}N_{reg}+N_{input}+N_{output}+2{\cdot}N+2$.

The complexity of solving a convex quadratic program with linear constraints depends on the number of Newton steps required to compute a feasible solution. In our implementation, we adopted LOQO [13] as the quadratic program solver. Generally speaking, LOQO finds an optimal solution for a convex quadratic program within 200 steps.

Floating-Node Replication

Min-cut, min-delay replication problem is to determine a set of nodes to replicate between two parts so as to minimize the cut size $|\vartheta|$ and satisfy the timing constraints. It is possible that TPIQ may not have a feasible final solution (i.e. $x_i \in \{0, 1\}$ $\forall$ $x_i \in \bar{X}$). The solution of TPIQ can differentiate the nodes which violate the timing constraints from those which do not. After all the nodes are anchored, some nodes may not have solutions of 1 or 0 because of the timing constraints. The nodes whose final solutions are between (0, 1) are called floating nodes.

Consider Figure 1-a where the floating node n_i receives inputs from both parts and sends its output to both parts. Let the solid and the dashed arrows indicate two critical timing path. If we move n_i to either part, one of the critical paths is cut twice. If we replicate this node as shown in Figure 1-b, then each critical path is cut only once.

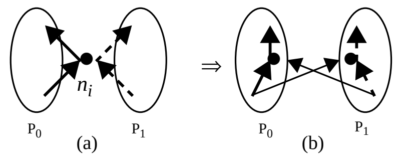

Figure 1: Floating-node replication.

We also use floating-node *expansion* or *reduction* to resolve timing violations. Details are omitted due to space limitation.

5. Experimental Results

The simulation was executed based on the unit delay model [7]. This delay model assumes that the external delay is one and ignores the internal delay and gate delay. The simulation assumed all nodes have unit size. The area balance criterion is 45% to 55% before node replication. Choosing the range of neighborhood regions is an accuracy/run-time trade-off. We chose R_0=[0, 0.2], and R_1=[0.8, 1].

The algorithm was tested on ISCAS89 benchmarks [14]. Table 1 is the comparison of the results of TPIQ phase I to other algorithms.The cycle time Φ_G is calculated as the maximum number of cut edges along paths from PIs (PPIs) to POs (PPOs). The results for the FM algorithm are reported as the best run from 200 runs. The results of TPIQ phase I are reported in column 2 which includes the cut size $|\vartheta|$ and the longest path delay Φ_G in parenthesis. Column 3 is the minimum $|\vartheta|$ among the 200 FM runs and its corresponding Φ_G in parenthesis. Column 4 is the minimum Φ_G among the 200 FM runs and its corresponding $|\vartheta|$ in parenthesis. We can see that phase I TPIQ is always better than FM algorithm in cut size. Column 5, 6 are the cut size results of CLIP [5] and ML_c [4] respectively. TPIQ phase I is superior to CLIP by 16.9% on the average and inferior to ML_c by 6.8%. The real advantage of TPIQ is however in its ability to minimize the cut size subject to given timing constraints as will be illustrated in Table 2. The other algorithms cannot perform this task.

Table 1: Results of TPIQ phase I and other algorithms.

benchmark	TPIQ phase I	min. $\|\vartheta\|$ of FM	min. Φ_G of FM	$\|\vartheta\|$ of CLIP	$\|\vartheta\|$ of ML_c
s1196	39(4)	39(4)	3(44)	40	39
s1238	40(5)	40(5)	3(47)	40	40
s1423	13(2)	13(2)	2(13)	12	12
s1488	42(3)	45(4)	3(48)	41	42
s1494	42(4)	45(4)	3(47)	42	41
s5378	58(3)	82(4)	3(101)	60	56
s9234.1	47(3)	51(3)	2(59)	45	40
s13207.1	65(5)	92(3)	2(96)	85	56
s15850.1	43(3)	101(3)	3(101)	81	42
s35932	42(1)	118(3)	2(132)	67	42
s38417	63(3)	127(5)	3(136)	75	50
s38584.1	50(3)	59(3)	3(59)	48	47
Sum of $\|\vartheta\|$	544	812	883	636	507
Improved	-	49.2%	-	16.9%	-6.8%

Table 2 lists the results of TPIQ phase II under different Φ_G constraints. All the arrival times for PIs (PPIs) are assumed to be 0 and the required times for POs (PPOs) are set to be Φ_G. The values outside the parentheses denote the number of replicated nodes, and the values inside parentheses denote the cut size. For any cycle time $\Phi_G \geq 2$, the number of replicated nodes is negligible. When Φ_G is as low as 2, the average cut size increases by 24% compared to that of TPIQ phase I partitioning solution, but the delay is improved by 42% When Φ_G is reduced to 1, the number of replicated nodes and cut size increase rapidly. Although Φ_G is improved by 69%, the replication rate increases to 4.7% and the average cut size increases by 344%. From the results, we see that setting Φ_G to 2 is a practical step under the unit delay model.

Table 2: Results of TPIQ phase II for different Φ_G under the unit delay model.

benchmark	Φ_G=1	Φ_G=2	Φ_G=3	Φ_G=4	Φ_G=5
s1196	78(193)	0(62)	0(40)	0(39)	
s1238	63(195)	0(63)	0(46)	0(46)	0(40)
s1423	42(132)	0(13)			
s1488	98(221)	0(51)	0(42)		
s1494	26(87)	0(47)	0(42)	0(42)	
s5378	10(119)	0(63)	0(58)		
s9234.1	56(184)	0(57)	0(47)		
s13207.1	75(283)	0(101)	0(95)	0(69)	0(65)
s15850.1	51(154)	0(43)	0(43)		
s35932	0(42)				
s38417	21(148)	0(87)	0(63)		
s38584.1	70(265)	0(59)	0(50)		

We have also generated results for a general delay model (D=1, d=$\delta(v_i)$=0.01, normalized required time of POs=2.5). In this case phase II TPIQ gives an average of 17.2% reduction in Φ_G at the cost of 1.2% node duplication. Tables are not included due to space limitation.

The run-times of TPIQ for ISCAS89 benchmarks measured on a SUN Sparc station are listed in Table 3. The run-time strongly depends on the circuit structures.

Table 3: Run-time of ISCAS89 benchmarks (unit: second).

benchmark	# nodes	phase I	phase II (Φ_G=1)	Total
s1196	575	395	15	410
s1238	554	383	15	398
s1423	753	54	13	67
s1488	686	434	10	444
s1494	680	391	1271	1662
s5378	3042	757	3439	4196
s9234.1	5883	513	1004	1517
s13207.1	8803	586	901	1487
s15850.1	10533	632	6232	6864
s35932	18148	867	-	867
s38417	23949	1752	7209	8961
s38584.1	21021	1433	10600	12033

6. Conclusion

We proposed a timing-driven partitioning algorithm TPIQ. The results of TPIQ demonstrates that by incorporating timing constraints into the partitioning and using replication algorithm we can achieve a min-cut partitioning solution which satisfies the delay target for the circuit.

Currently, we are using TPIQ as the partitioner of a timing-driven placement system. By restricting the number of cuts of critical paths, we hope to achieve the delay minimization.

Acknowledgments

The authors would like to thank Dr. Charles Alpert for providing us the simulation results of CLIP and ML_c.

References

[1] C. M. Fiduccia and R. M. Mattheyses. “A Linear Time Heuristic for Improving Network Partition”. In *19th DAC*, pages 175-181, 1982.

[2] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu. “Large Scale Circuit Partitioning With Loose/Stable Net Removal And Signal Flow Based Clustering”. In *ICCAD*, pages 441-446, 1997.

[3] S. Dutt, W. Deng. “A Probability-based Approach to VLSI Circuit Partitioning”. In *33th DAC*, pages 100-105, 1996.

[4] C. J. Alpert, J. H. Huang, and A. B. Kahng. “Multilevel Circuit Partitioning”. In *34th DAC*, pages 530-533, 1997.

[5] S.Dutt and W. Deng. “VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques”. In *33th DAC*, pages 194-200, 1996.

[6] A. B. Kahng. “Features for Partitioning in Physic Design”. In *ISPD*, pages 190-193, 1998.

[7] E. L. Lawler, K. N. Levitt, and J. Turner. “Module Clustering to Minimize Delay in Digital Network”. *IEEE Trans. on Computers*, pages 47-57, 1969.

[8] R. Murgai, R. K. Brayton, and A. S. Vincentelli. “On Clustering for Minimum Delay/Area”. In *ICCAD*, pages 6-9, 1991.

[9] H. Yang and D. F. Wong. “Circuit Clustering for Delay Minimization Under Area and Pin Constraints”. In *Proc. of the International Workshop on FPGA*, 1994.

[10] R. Kuznar, Doctoral Thesis, Univ. of Liubljana, 1996.

[11] M. Shih and E. S. Kuh. “Quadratic boolean programming for performance-driven system partitioning”. In *DAC*, pages 761-765, 1993.

[12] R. B. Hitchcock, G. L. Smith, and D. D. Cheng. “Timing Analysis of Computer Hardware”. *IBM Journal of Research and Development*, 26(1), pages 100-105, 1983.

[13] R. J. Vanderbei. “LOQO: An Interior Point Code for Quadratic Programming”, Technical Report, Princeton University, 1995.

[14] ISCAS89 benchmarks. at “http://www.cbl.ncsu.edu/www/”.