Timing-driven Placement Based on Partitioning with Dynamic Cut-net Control[*]

Shih-Lian Ou and Massoud Pedram
Department of EE-Systems, University of Southern California
Los Angeles, CA90089

Abstract
This paper presents a partitioning-based, timing-driven placement algorithm. The partitioning step itself is timing-driven and based on solving a quadratic programming problem iteratively. The placement algorithm does not rely on interleaved timing calculations, which tend to be inaccurate. Instead, it achieves the desired result by controlling the number of times that a path in the circuit can be cut. In addition to the cutting constraint, a pre-locking mechanism and timing-aware terminal propagation are developed and integrated into the flow. The detailed placement step is formulated as a constrained quadratic program and solved efficiently. Results show improvements of 23.41% on average compared to another timing-driven placement system Timing-QUAD and significant improvements over Eisenmann's placement algorithm.

1. Introduction

Steady advances in semiconductor technology have resulted in significant improvements in the performance of CMOS devices. For these improvements to translate to the VLSI circuit and system designs, one must resort to powerful, timing-aware computer aided design techniques and tools at all levels of design abstraction, including architecture logic and physical level. A good placement tool must be capable of minimizing the maximum circuit delay by reducing the delays of interconnects along the timing critical paths. This is a challenging task because of the large number of paths that must be considered and the complex inter-dependence among the various paths.

Timing-driven placement schemes have been classified as either net-based [1][2][3][4] or path-based [5][6][7]. Net-based algorithms seek to control the delay on a signal path by imposing a delay upper-bound or by assigning a weight to each net. The net-based approach usually overconstrains the placement algorithm. Path-based approaches model the problem correctly, but are only applicable to small circuits since explicit enumeration of all paths is an intractable task. As a result, most of the timing-driven placement algorithms proposed to-date are net-based [8][9][10]. More precisely, they perform circuit placement by interleaving a timing-analysis step with a placement step. The timing-analysis step is responsible for identifying the critical nets. The weights of the circuit nets are updated based on their timing criticality. The placement step then determines the relative locations of gates according to the net weights. These two steps are interleaved until the target delay is achieved or the current result cannot be further improved. The timing analysis techniques used in this two-step flow are standard. The methods used for identifying the critical nets and the placement technique that vary from one implementation to next.

The authors of [8] adopt a quadratic objective function formulation for minimizing the wire length. They also perform a longest path calculation during each timing analysis step. The weights of nets along the critical path are updated by adding a constant to the corresponding edge weights. This approach however suffers from some major weaknesses. The first drawback is the so-called oscillation problem. This is a commonly-encountered problem in circuits where many paths have nearly the same negative slack. In addition to the oscillation problem, quadratic formulation does not minimize the length of nets, instead, it minimizes the square of edge lengths. As a result, it may result in a number of medium-length nets in order to eliminate one long net. Sometimes however a long net may be a better solution than a number of medium-length nets, for example, when the paths which share this long net have very small intrinsic delays.

The authors of [10] implement the timing-driven placement by repeatedly performing quadrisectioning on the circuit. The top-down quadrisectioning is driven by net cost vectors which is updated during the timing-analysis steps. Top-down partitioning-based timing-driven placements avoid the problems mentioned above. However, the timing information available during the partitioning stages is incomplete until the final placement layout is reached. A partitioning-based placement may thus make irreversible, incorrect decisions during early partitioning stages due to the incomplete timing information.

A timing-driven placement algorithm based on simulated annealing is presented in [11]. It is a path-based approach. However, because it is impossible to enumerate all the paths, the authors of [11] compute the first twenty longest paths of the circuit. The difference of user-specified upper bounds and critical path delays are added to the cost function as a timing penalty. However, in many cases, the path delay of the most critical paths may differ from the delays of hundreds or thousands of the other paths in the circuit by only a small amount. Considering a small number of such paths is not very helpful. The ideal situation is to take all such nearly-critical paths into account simultaneously. This however significantly increases the computational complexity.

In this paper, we present a partitioning-based timing-driven placement algorithm TPIQ_P which is implemented based on the principle that the number of cuts on paths in the circuits must be carefully controlled. A path which is cut earlier has a higher probability to assume a longer wire length. So a critical path must either not be cut in the early partitioning stages or restricted to have fewer cuts in the subsequent partitioning stages. To achieve this goal, simply updating the weights of critical nets is not of much help since there is no way to estimate the net weights exactly. Controlling the number of cuts on paths in any given partitioning stage is also not helpful because a net may be cut several times during different partitioning stages. Our approach borrows from both net-based and path-based algorithms. More precisely, critical nets are given large weights exactly like what the net-based algorithms do. Paths are also given an upper limit on the maximum number of cuts they can tolerate. Imposing this limit on a path helps achieve the goal of making the path as geometrically "straight" as possible. Note that when the number of cuts along a path is decreased, the path delay is reduced. Our experimental results on a number of benchmark circuits show that TPIQ_P outperforms the best published results of any timing-driven placement algorithm.

The remainder of the paper is organized as follows: Section 2 describes of our timing-driven placement. Section 3 is the formulation. The experimental results are given in section 4. Section 5 is the conclusion.

2. Description

Partitioning-based placement determines the locations of gates on the layout area by repeatedly partitioning the given circuit into two or more sub-circuits. The available layout area is also partitioned into subsections. Each of the sub-circuits is assigned to a subsection. This process is carried out until each sub-circuits consists of a small number of cells as Figure 1 shows.

Traditionally, the objective of dividing a given circuit in a partitioning-based placement algorithm is to minimize the number of cut nets among sub-circuits. However, min-cut is a poor model of the real placement objective, especially because it ignores the circuit delay. Although static timing analysis can help calculate the critical path delays, this timing information is not accurate unless the final partitioning step is executed. Because of this intrinsic inaccuracy, in TPIQ_P, we do not rely on performing timing recalculation after each partitioning step. Instead, we predict the critical paths from the beginning and then enforce cutting constraints (i.e.,

[*]This work was supported in part by SRC under contract No. 98-DJ-606 and NSF under contract No. MIP-9628999.

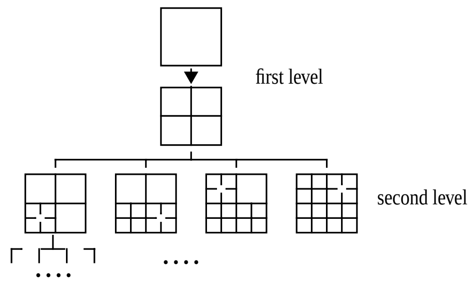

Figure 1. The layout area of a 4-way partitioning-based placement.

maximum number of times the path can be cut during the iterative partitioning steps) on these paths.

As recursive partitioning steps take place, critical paths in the circuit change. Intrinsic delays are fixed throughout the whole partitioning steps. It is the extrinsic edge delays that change and cause non-critical paths to become critical or vise versa. The main task of a timing-driven placement is to control the lengths of nets along the critical paths.

In our approach, the extrinsic delay is calculated from a lumped Elmore delay model [12].

Let's consider the path which has the longest intrinsic delay first. The path with maximum logic delay (sum of intrinsic and extrinsic delays when $R_{net} = C_{net} = 0$) is obtained by performing a timing analysis. Let the logic delay of that path be denoted as $d_{longest}$. The remaining M-1 longest paths may have logic delays very close to $d_{longest}$. Minimizing only one longest path may cause another long path to become more critical immediately. Hence, it is necessary to take the M longest paths into account simultaneously. These M longest paths are defined as the critical path set η in TPIQ_P. A path L which is not in the critical path set η may become critical if it contains either a large net or receives many cut-nets. A large net tends to have longer net length and, thus contributes more to the path delay. Similarly, a path that has already been cut a large number of times has a high probability of becoming critical if the number of future cuts is not controlled. To prevent such nets from generating new critical paths, the large nets and the cut nets must also be considered as critical nets.

Here we describe the critical net set. Let Γ denote the critical net set, n_k the kth net, $|n_k|$ the number of nodes that n_k connects, and w_k the weight of net n_k. Initially, every net has a unit weight. $\beta(n_k)$ is a flag that indicates whether n_k has been cut or not.

$$\beta(n_k) = \begin{cases} 1 & if\ \ n_k\ \ has\ \ been\ \ cut \\ 0 & if\ \ n_k\ \ has\ \ not\ \ been\ \ cut \end{cases}$$

We define the critical net set Γ as the union of nets in η, large nets, and cut nets.

$$\Gamma = \{n_k | \ n_k \in \text{ nets in } \eta\} \cup \{n_k | \ |n_k| > \text{size threshold}\} \cup \{n_k | \ \beta(n_k) = 1\},$$

where the size threshold is a user-specified value. All critical nets in a part p_i are assigned the same weight W_i. W_i must be large enough so that the cells of a critical net are forced to remain in the same part. We set W_i to be equal to the summation of the weights of non-critical nets in a part p_i.

$$W_i = \sum_{n_k \in p_i} (1 - \beta(n_k)) \cdot w_k$$

If a non-critical net is cut in the current partitioning stage, it becomes critical in the remaining partitioning stages and its weight is updated. Appending the cut nets to the critical net set prevents the cut nets from being cut further and only allows the uncut, non-critical nets to be possibly cut in the subsequent partitioning stages.

Concentrating on handling critical nets only does not ensure a reduction in circuit delay. The delay of a non-critical path may increase rapidly if it is cut many times during an early partitioning stage. So, it is necessary to set an upper-bound on the number of allowed cuts on each path. We adopt TPIQ [13] as the partitioner for the placement tool TPIQ_P to achieve this goal.

We illustrate the processing procedure of TPIQ_P in Figure 2. Before the partitioning stage begins, the set of paths which have longer logic delays are identified. The critical nets are thus determined and the corresponding weights are updated. The updated netlist is passed as input to a partitioner whose responsibility is to minimize the cut-size. A simple calculation follows to check which nodes violate the cutting poicy. The cutting policy is a set of con-

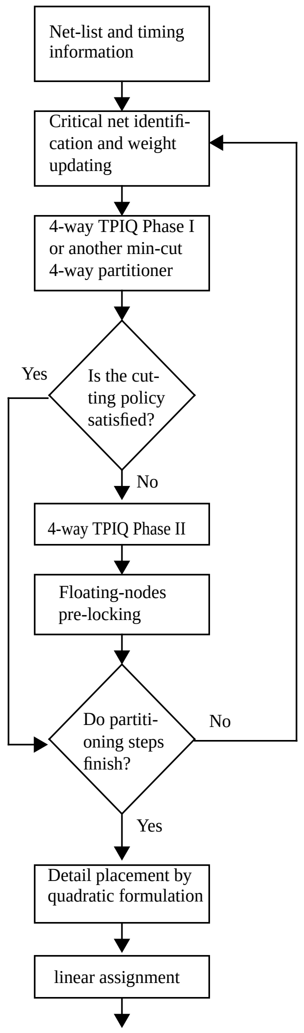
Net-list and timing information
Critical net identification and weight updating
4-way TPIQ Phase I or another min-cut 4-way partitioner
Is the cutting policy satisfied?
Yes
No
4-way TPIQ Phase II
Floating-nodes pre-locking
Do partitioning steps finish?
No
Yes
Detail placement by quadratic formulation
linear assignment

Figure 2. Flow of TPIQ_P.

straints which restrict the number of cuts allowed for each path. The cutting policy is comprised of the restrictions for the paths in the circuits. Because an output node can be the output of both critical paths and non-critical paths, the constraints for general paths and the constraints for critical paths must be formulated separately. Thus, the cutting policy constraints consist of two categories, one for general paths and the other for critical paths. Let PC denote the cutting policy. $PC=\{PC_G, PC_L\}$ where PC_L denotes the maximum number of cuts allowed for a critical path L and PC_G refers to the maximum number of cuts allowed for a general path.

In general, PC_L is more strict (i.e. smaller) than PC_G. If there are paths whose cuts exceed the related PCs, TPIQ Phase II is applied to satisfy the cutting policy while trying to minimize the cut-size.

We have extended the 2-way partitioning of TPIQ to a quadrisection partitioning to reflect the two dimensional characteristics of the circuit placement. The solution of a quadratic programming in TPIQ_P is thus represented by an (X, Y) pair, where $X = [x_1, x_2, ... ,x_N]^T$ and $Y = [Y_1, Y_2, ..., y_N]^T$. Four dummy fixed nodes are introduced, one for each quadrant. These fixed nodes are placed at (0, 0), (1, 0), (0, 1), and (1, 1) respectively. Figure 3 shows the parts P_0, P_1, P_2, and P_3 and the corresponding (x, y) pairs of fixed nodes.

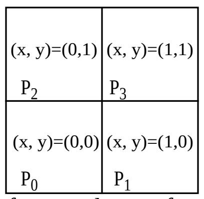

Figure 3. The four quadrants of a quadrisection and their (x, y) encodings.

As TPIQ phase II proceeds, nodes are gradually anchored to one of the fixed nodes. If the final solution of TPIQ Phase II contains values that do not converge to 0 or 1 in either x-coordinate or y-coordinate, the corresponding nodes will be pre-locked to the part which contains the corresponding nodes. Pre-locking a node to a specific sub-part prevents subsequent partitioning steps from generating a longer path delay for paths that go through the pre-locked node (please see below for detail).The partitioning and pre-locking procedure are iteratively executed until the numbers of nodes in all parts are below a threshold value τ.

The position of a node v_i is represented by two position vectors V_i^x and V_i^y:

$V_i^x = [v_1^{xi}, v_2^{xi}, \ldots v_m^{xi}]$, $v_j^{xi} \in \{0, 1\}$ for $1 \le j \le m$.

where m is the maximum level of the partitioning process. Similarly,

$V_i^y = [v_1^{yi}, v_2^{yi}, \ldots v_m^{yi}]$, $v_j^{yi} \in \{0, 1\}$ for $1 \le j \le m$.

The pair (v_j^{xi}, v_j^{yi}) indicates the coordinates of the part where node v_i is assigned to after the j_{th} level of the partitioning process. The value of v_j^{xi} (or v_j^{yi}) is determined by either the partitioning process itself or by the pre-locking procedure. The solution pair (x_i, y_i) of TPIQ Phase II indicates which part v_i should be assigned to. If both x_i and y_i converge to binary values 0 or 1, the part is immediately determined. Otherwise, v_i is a floating node. A floating node is critical to the paths which come from (or go to) different parts and pass through that node. Replicating floating nodes as TPIQ does, is not practical in this application because the cost of replicating nodes is too high for placement (also, the node replication may not be allowed). Instead of replicating nodes, we pre-lock the floating nodes to specific parts for the remaining partitioning stages. Figure 4 depicts the assignments for floating nodes that have different connections to the nodes in different parts. Notice that if a node v_j is assigned to P_0 at the i_{th} partitioning level, its coordinates (v_i^{xj}, v_i^{yj}) will be (0, 0). Similarly a P_1 assignment implies (v_i^{xj}, v_i^{yj}) will be (1, 0) and so on. Shaded grids are the sub-parts to which the floating nodes must be fixed during the remaining partitioning stages. A floating node may have more than one choice of sub-part to be fixed to. The selection of sub-parts is then based on the criterion of maintaining the size balance.To better understand the purpose of pre-locking, we provide an example which is depicted in Figure 5. Suppose that after the first three levels of partitioning, node v_i has been assigned to the part that is shown in Figure 6. At that time, the position vectors of v_i are $V_i^x = [0, 1, 1, -, -, \ldots, -]$ and $V_i^y = [1, 0, 0, -, -, \ldots, -]$, where "-" denotes an indeterminate value. If v_i is a floating node and is required to be assigned to the center of the whole layout area, v_i is always fixed to sub-part P_1 in the remaining partitioning stages. The pre-locking procedure will therefore set V_i^x to [0, 1, 1, 1, 1, ..., 1] and V_i^y to [1, 0, 0, 0, 0,..., 0] before the next partitioning stage begins.

The detailed placement after the partitioning stages is determined by solving a quadratic optimization procedure as will be detailed later.

3. Formulation

The objective function $f(X, Y)$ for the partitioning process is modified by mapping the variables in TPIQ from one-dimensional space to two-dimensional space:

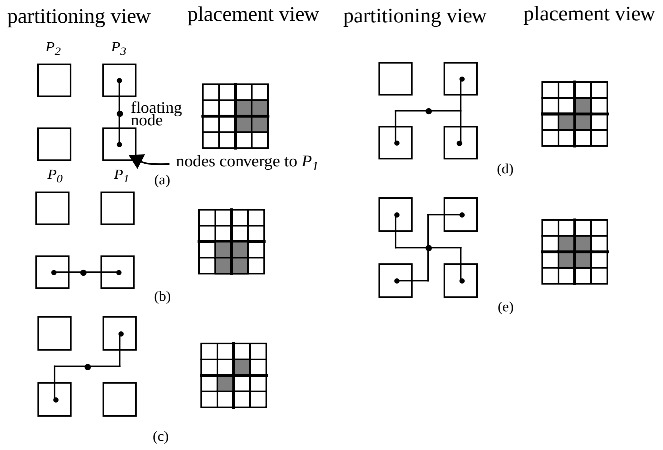

Figure 4. Illustration of pre-locking mechanism.

$$\frac{1}{2}\sum_{x_i,\, x_j \in X} c_{ij}^x (x_i - x_j)^2 + \frac{1}{2}\sum_{y_i,\, y_j \in Y} c_{ij}^y (y_i - y_j)^2$$
$$+ c_{max}^x\left[\sum_{v_i \in P_0, P_2} (x_i - 0) + \sum_{v_i \in P_1, P_3} (1 - x_i)\right]$$
$$+ c_{max}^y\left[\sum_{v_i \in P_0, P_1} (y_i - 0) + \sum_{v_i \in P_2, P_3} (1 - y_i)\right],$$
$$0 \le x_i \le 1,\ 0 \le y_i \le 1,\ x_i \in X, y_i \in Y.$$

c_{ij}^x and c_{ij}^y are the edge weights for the edge (v_i, v_j) in x-coordinate and y-coordinate respectively. c_{max}^x and c_{max}^y are the edge weights between a circuit node and a fixed dummy node when the circuit node is anchored to that fixed node. The method for calculating the edge weights is the same as in [13].

The constraints are formulated as follows.

Cutting Policy Constraints

These constraints attempt to limit from above the number of times a path is cut.

For every node in the circuit, we formulate the timing relation by using the block-oriented timing analysis algorithm of [14]. The arrival time b_i is defined as the maximum number of cuts for all paths from a primary input (or an output of a register) to the node v_i.

$b_i + |x_j - x_i| + |y_j - y_i| \le b_j$, $\forall\ (v_i, v_j) \in E$,

$b_i = 0$, if $v_i \in$ {primary inputs or output of registers},

$b_j = PC_G$, if $v_j \in$ {primary outputs or input of registers},

where PC_G is a user-defined value which specifies the maximum number of cuts allowed for paths from any input node to the output node v_j.

The cutting policy for a critical path L is formulated as

$$\sum_{(v_i,\, v_j) \in L} |x_i - x_j| + |y_i - y_j| \le PC_L \quad , \forall \text{ path } L \in \eta.$$

The PC_L of a critical path depends on the I/O pin positions of that path, which is specified by the user. Also note that the left hand side of the above inequality is not necessary an integer. It will however converge to an integer as the optimization process progresses.

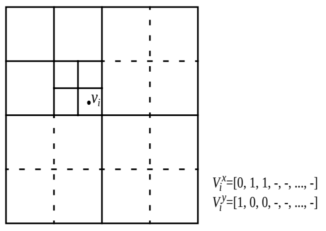
v_i
V_i^x=[0, 1, 1, -, -, ..., -]
V_i^y=[1, 0, 0, -, -, ..., -]

Figure 5. Explanation of the position vector.

Capacity Constraints

Let $s(v_i)$ denote the size of node v_i. $S_k(v_i) = \Sigma_k s(v_i)$ denotes the summation of node sizes in a sub-part $p_k \in \{P_0, P_1, P_2, P_3\}$. Then we have the following capacity constraints:

Lower bound $\le S_k(v_i) \le$ Upper bound, $\forall\ p_k \in \{P_0, P_1, P_2, P_3\}$.

Binary Boundary Constraints

$0 \le x_i \le 1$, $0 \le y_i \le 1$, $\forall\ x_i \in X$ and $y_i \in Y$.

Pre-locking Constraints

In addition to the primary input nodes and the primary output nodes whose pin locations are specified, the position vector V_i^x or V_i^y of some floating nodes have to be determined and locked. The pre-locked values in V_i^x and V_i^y of primary I/O nodes and floating nodes are transformed into constraints by setting

$x_i = 1$ (or 0) if v_i is pre-locked to sub-parts P_1 or P_3 (P_0 or P_2), or it is fixed to the right (or left) side of a part,

$y_i = 1$ (or 0) if v_i is pre-locked to sub-parts P_2 or P_3 (P_0 or P_1), or it is fixed to the top (or bottom) of a part,

$x_i = 1$ (or 0) and $y_i = 1$ (or 0) if v_i is pre-locked to a specific part.

Detailed Placement

The final implementation of the partitioning process is a global allocation of circuit components. To assign the circuit components to detailed positions, we first calculate the coordinates of every part p_k in the whole layout area. Let the range of x-coordinates of p_k be denoted by $[x_L^k, x_R^k]$ and the range of y-coordinates be denoted by $[y_B^k, y_U^k]$. The detailed locations of circuit components are determined by solving the following quadratic formulation:

minimize $f(X, Y) = X^T QX + Y^T QY$

subject to

1. center of mass constraints

$$\frac{1}{|p_k|}\sum_{x_i \in p_k} x_i = \overline{x_n},$$

$$\frac{1}{|p_k|}\sum_{y_i \in p_k} y_i = \overline{y_n}$$, for every node $v_i \in p_k$.

where $(\overline{x_n}, \overline{y_n})$ is the center of mass of a part p_k.

$$\overline{x_n} = \frac{x_L^k + x_R^k}{2},\ \overline{y_n} = \frac{y_B^k + y_U^k}{2}.$$

2. boundary constraints

$x_L^k \le x_i \le x_R^k$, $y_B^k \le y_i \le y_U^k$

3. pre-locking constraints

$x_L^k \le x_i \le x_L^k + \mu$ if v_i is pre-locked to left side of part p_k (sub-parts P_0 or P_2 of p_k).

$y_B^k \le y_i \le y_B^k + \mu$ if v_i is pre-locked to bottom side of part p_k (sub-parts P_0 or P_1 of p_k).

$x_R^k - \mu \le x_i \le x_R^k$ if v_i is pre-locked to right side of part p_k (sub-parts P_1 or P_3 of p_k).

$y_U^k - \mu \le y_i \le y_U^k$ if v_i is pre-locked to top side of part p_k (sub-parts P_2 or P_3 of p_k).

where μ is a user-specified values.

The adjacency matrix Q in the objective function is obtained by setting the edge weight of a net n_j as $1/|n_j|$. A linear assignment is applied row by row to eliminate the overlap based on the solutions of the quadratic programming.

Terminal Propagation

Terminal propagation is an important operation in all partitioning-based placement algorithms because it allows for the effect of placing components in same part to influence the placement of components in the subsequently processed parts. A terminal propagation solution obtained at the earlier partitioning stage tends to bias the solution of the succeeding partitioning stages. The traditional terminal propagation deals with the problem of wire-length minimization only. It does not consider the problem of delay minimization. Terminal propagation without considering path delays may however result in a longer circuit delay in the quadrisection placement context. In TPIQ_P, the positions of terminal propagation pins are calculated immediately after a partitioning step is finished. If a critical path $L \in \eta$ is cut, we identify the nodes which belong to the path L. We then compute the center of mass of these nodes and let the terminal propagation pins be locked to a sub-part which is close to this center. To better illustrate the process, let's consider the example in Figure 6. In Figure 7-a, a partitioning step assigns nodes a, b, and c to parts P_3, P_1, and P_0, respectively. Since the critical path <*a-b-c*> is cut, we calculate the center of mass of these three nodes, which happens to be the center of the chip. Now consider terminal propagation of node a. This node is placed in part P_3 which itself has four sub-parts P_0, P_1, P_2 and P_3. Sub-part P_0 of part P_3 is closest to the center of mass of <*a-b-c*>, hence node a will be propagated to the middle of the bottom face of sub-part P_0 of part P_3. Similarly, node c will be propagated to the middle of the right face of sub-part P_3 of part P_0. In fact, we pre-lock the terminal propagation pin of (a, b) and the terminal propagation pin of (b, c) to subparts P_0 and P_3, respectively in all subsequent partitioning stages.

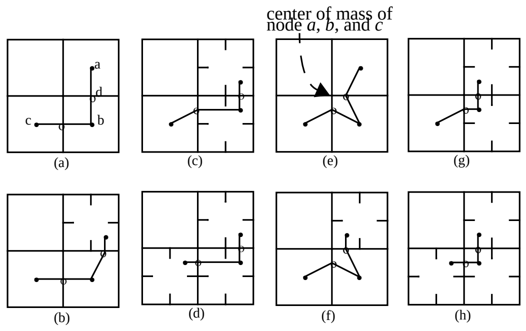

Figure 6. Terminal propagation in non-timing-driven placement approach (a)-(d) and in TPIQ_P (e)-(h).

4. Experimental Results

TPIQ_P was implemented on a Sun Ultra Sparc II machine. In our experimental simulation, we first ran timing analysis and sought out the first 250 longest paths by using Dreyfus's method [15] to produce the longest path set. Then the paths whose path delays were smaller than $0.9d_{longest}$ were deleted from the longest path set. All nets in the critical path set along with those nets whose fanout count exceeded 30 pins formed the initial critical net set. As the partitioning stages proceeded, the newest cut nets were added into the critical net set. TPIQ Phase I was replaced by hMeTis [16] in the simulation to reduce the run time. We adopted the same parameters used by Timing-QUAD [10] which is a partitioning-based timing-driven placement system. We ignored nets which have 200 or more output pins. From our experience with TPIQ, setting PC_G to 2 for general paths in sequential circuits is a good choice for circuit bipartitioning. Similarly, a reasonable PC_G is 4 for the quadrisection partitioning. Our experiments showed that when PC_G was decreased to 3, a large number of floating nodes were generated and the size balance between parts became difficult to maintain. The cutting policy for critical paths was the same for all the benchmarks. The maximum number of edge-cuts allowed for a critical path depends on its I/O pin positions. If both the input and output pins are fixed on the top or bottom side, we set PC_L of the path to 3. If one I/O pin is on top and one is on bottom, then PC_L=1. For other situations, we used PC_L=2. The capacitance per wire length is 242 pF/m and the resistance per wire length is 25.5kΩ/m in our simulation. The threshold value τ was 64 and μ is 0.1.

Table 1 shows results of comparing our approach with Timing-QUAD. The values in parentheses are the longest logic path delay in the circuit. Because there may have some differences between

Timing-QUAD and TPIQ_P in transforming benchmarks, the maximum logic path delays were somewhat different. However, the difference does not affect the comparison. From the table we can see that our approach has smaller circuit delays for all tested benchmarks. Although TPIQ_P does not run timing analysis iteratively, it still outperforms Timing-QUAD by an average of 23.41% in terms of circuit delay. The runtimes of TPIQ_P are reasonable, for example, the runtime for *biomed* is 2016 second and 12970 second for *avq.small* on a Sun Ultra Sparc II.

Table 1: Comparison of circuit delay for TPIQ_P and Timing-QUAD

Benchmark	# cell	# net	Timing-QUAD	TPIQ_P	Impr.
fract	125	147	18.4 (10.6)	11.91 (11.56)	35.27%
struct	1888	1920	79.3 (40.0)	55.51 (48.71)	30.00%
biomed	6417	5742	29.3 -	20.27 (14.20)	30.82%
avq.small	21854	22124	71.0 (37.3)	59.65 (36.85)	15.99%
avq_large	25114	25384	76.9 -	73.07 (46.44)	4.98%
Average					23.41%

We also report the results of TPIQ_P for different PC_G's in Table 2. The PC_G in column 5 is ∞, which means that we updated the weights of critical nets but did not run TPIQ Phase II in this case. As we can see, smaller PC_G always results in better performance. However, PC_G cannot be reduced indefinitely. A value of 4 appears to be reasonable.

Table 2: Comparison of different upper bound on cuts for TPIQ_P

Benchmark	PC_G= 4	PC_G = 8	PC_G = 10	$PC_G = \infty$
fract Impr. to PC_G=4	11.91 -	12.77 6.73%	12.77 6.73%	12.77 6.73%
struct Impr. to PC_G=4	55.51 -	55.60 0.16%	55.61 0.17%	55.69 0.32%
biomed Impr. to PC_G=4	20.27 -	20.52 1.22%	20.53 1.26%	21.11 3.98%
avq.small Impr. to PC_G=4	59.65 -	61.98 3.76%	66.19 9.88%	71.21 16.23%
avq_large Impr. to PC_G=4	73.07 -	74.19 1.51%	78.32 6.70%	89.24 18.11%
Avg. in Impr.	-	2.68%	4.95%	9.08%

Table 3: Comparison of TPIQ_P with Eisenmann's method

Benchmark	*E* of Eisenmann's method	*E* of TPIQ_P
fract	51%	93%
struct	28%	32%
biomed	59%	87%
avq.small	68%	82%
avq_large	57%	87%
Average	53%	76%

We also compare the performance of our approach (PC_G=4) to that of Eisenmann's method [9]. In order to coincide with the delay calculation method used in [9], we measured the net length with Steiner tree as [9] did and calculate the exploitation rate *E* which is defined as follows:

E = (delay without timing-driven feature - delay with timing-driven feature) / (delay without timing-driven feature - longest intrinsic path delay feature)

The exploitation rate *E* indicates how close the results of a timing-driven placement is to its the best possible solution for the placement problem. The comparisons are shown in Table 3. TPIQ_P is superior to Eisenmann's method in all testing cases.

5. Conclusion

We presented TPIQ_P, a new timing-driven placement algorithm based on iterative timing-driven partitioning with dynamic control of the number of times a set of potentially critical paths are cut. The proposed algorithm does not rely on interleaved timing calculations which tend to be inaccurate. Instead it maintains timing-aware placement by paying attention to how many times a path is cut by the cut-lines. Experimental results showed the effectiveness of the proposed approach in improving the delays of MCNC benchmarks compared to the other timing-driven placement algorithms.

Future work will focus on incorporation of partitioning algorithms that account for other objectives such as congestion and routability in addition to timing. This is specially important during the early partitioning steps. The later partitioning may remain as they are now.

Reference

[1] A. E Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip Layout Optimization using Critical Path Weighting," *Proc. 21st ACM/IEEE Design Automation Conference*, pages 133-136, 1984.

[2] M. Burstein and M. N. Youssef, "Timing Influenced Layout Design," *Proc. 22nd ACM/IEEE Design Automation Conference*, pages 124-130, 1985.

[3] R. S. Tsay and J. Koehl, "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement," In *Proc. 28th ACM/IEEE Design Automation Conference*, pages 636-639, 1991.

[4] P. Hauge, R. Nair, and E. Yoffa. "Circuit Placement for Predictable Performance," In *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pages 88-91, 1987.

[5] 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. 27th ACM/IEEE Design Automation Conference*, pages 84-89, 1990.

[6] M. A. B. Jackson and E. S. Kuh, "Performance-Driven Placement of Cell Based IC's," In *Proc. 26th ACM/IEEE Design Automation Conference*, pages 370-375, 1989.

[7] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A Performance Driven Placement Algorithm," *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 39, No. 11, pages 825-840, November 1992.

[8] B. M. Riess and G. G. Ettelt, "SPEED: Fast and Efficient Timing Driven Placement," In *Proceedings of the IEEE International Symposium on Circus and System*, pages 377-380, 1995.

[9] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," In *35th Proceedings of the ACM/IEEE Design Automation Conference*, pages 269-274, 1998.

[10] D. J. Huang and A. B. Kahng,. "Partitioning-Based Standard-Cell Global Placement with an Exact Objective," In *ACM/IEEE ISPD*, page 18-24, 1997.

[11] W. Swartz and C. Sechen. "Timing Driven Placement for Large Standard Cell Circuit," In *32nd Proceedings of the ACM/IEEE Design Automation Conference*, pages 211-215, 1995.

[12] J. Rubinstein, J. Paul Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Transactions on Computer-Aided Design*, vol. CAD-2, no. 3, pages 202-211, 1983.

[13] S. Ou and M. Pedram. "Timing-Driven Bipartitioning with Replication Using Iterative Quadratic Programming". *Proc. of Asia and South Pacific Design Automation Conference.* pages 105-108, Feb. 1999.

[14] 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.

[15] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operation Research*, Vol. 17, pages 395-412, 1969

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. "Multilevel Hypergraph Partitioning: Application in VLSI Domain," In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526-529, 1997.