Concurrent Logic Restructuring and Placement for Timing Closure*

Jinan Lou, Wei Chen, Massoud Pedram
Department of Electrical Engineering – Systems
University of Southern California, Los Angeles, CA 90089

ABSTRACT: In this paper, an algorithm for simultaneous logic restructuring and placement is presented. This algorithm first constructs a set of super-cells along the critical paths and then generates the set of non-inferior re-mapping solutions for each super-cell. The best mapping and placement solutions for all super-cells are obtained by solving a generalized geometric programming (GGP) problem. The process of identifying and optimizing the critical paths is iterated until timing closure is achieved. Experimental results on a set of *MCNC* benchmarks demonstrate the effectiveness of our algorithm.

I. Introduction

The ability to achieve timing closure in a complex circuit design is one of the most important features that designers are looking for in today's EDA tools. Timing corrections at postlayout level are often used to help achieve this goal. However, as feature sizes decrease to very deep submicron (VDSM) and clock frequencies increase to over 500MHz, simple techniques like in-place gate sizing and buffering are not able to produce satisfactory results. More powerful algorithms that consider additional optimization dimensions should be employed to deal with the VDSM-specific problems such as the increasing dominance of wire delays.

I.1. Background

Gate sizing has been discussed extensively in literature [3][7][8]. The work in [7] uses a discrete sizing model which is often too expensive to use even after the circuit is partitioned into smaller parts. The authors in [3][8] use continuous sizing methods, and formulate the sizing problem as a mathematical programming problem. In these works, the authors only adjust the gate sizes to match the output loads of the gates. Other optimization techniques such as gate placement and adjustment of the wire loads of the gates are not considered. Most recently, the authors of [5] proposed a method that performs simultaneous gate resizing and placement.

The algorithm presented in this paper extends previous works by exploring a larger portion of the optimization space. It simultaneously performs logic restructuring with repositioning of the gates in the k most-critical paths to improve the postlayout timing. Moreover, these optimizations are performed in a constraint-driven manner such that while improving the performance, the perturbation to the original design is minimized.

I.2. Delay Model and Timing Analysis

The delay model is given as:

$$d_{ij} = \tau_i + r \times (c_{net} + c_{load}) + r_{net} \times c_{load} \qquad (1)$$

where d_{ij} is the delay from pin i to pin j as shown in Figure 1, τ_i is the intrinsic delay of pin i, r is driver resistance, r_{net} and c_{net} are the resistance and capacitance for the net, and c_{load} is the summation of the input capacitances of the fanouts.

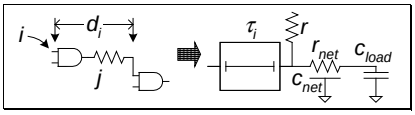

Figure 1. Delay Model.

Arrival times are calculated from the primary inputs to the primary outputs in the topological order. Required times are calculated in the reversed order. The arrival time of pin j of a gate is given as: a_j= MAX $\{(a_i+d_{ij} \mid i \in input(j)\}$, and the required time of pin i is given as: req_i=MIN$\{(req_j-d_{ij}) \mid j \in output(i)\}$. The slack can then be computed as $s_i=a_i-req_i$.

I.3. Wire Load Estimation

Wire load of a gate is estimated by the bounding box of its fanouts. The capacitance c_{net} and resistance r_{net} is given by:

$$c_{net} = \rho \cdot [C_{hor}(xnet_{\max} - xnet_{\min}) + C_{ver}(ynet_{\max} - ynet_{\min})]$$
$$r_{net} = \rho \cdot [R_{hor}(xnet_{\max} - xnet_{\min}) + R_{ver}(ynet_{\max} - ynet_{\min})] \qquad (2)$$

where $xnet_{max}$ and $xnet_{min}$ are the maximum and minimum x coordinates of the fanouts, $ynet_{max}$ and $ynet_{min}$ are the maximum and minimum y coordinates. C_{hor}, C_{ver}, R_{hor} and R_{ver} are the unit capacitance and unit resistance for the horizontal and vertical interconnects, respectively. ρ is a parameter to adjust the estimation error of the bounding box interconnect model [6]. For $n \le 10$, the values of ρ are produced in Table 1. We use the following equation for $n>10$:

$$\lim_{n->\infty} \rho = (\sqrt{n} + 1)/2 \qquad (3)$$

n	2	3	4	5	6	7	8	9	10
ρ	1	1	3/2	3/2	5/3	7/4	11/6	2	2

Table 1: Worst case equi-perimeter net lengths.

I.4. Generalized Geometric Programming

Definition 1: *Geometric Programming* (GP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *posynomials* [2].

A GP problem is given as follows:

minimize $p_0(x)$
s.t. $p_k(x) \le 0$, $k = 1, 2, \ldots m$
where $p_0, p_1, \ldots\ldots, p_m$ are posynomial functions.

Definition 2: *Generalized Geometric Programming* (GGP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *polynomials* [2].

A GGP problem is given as follows:

minimize $g_0(x)$
s.t. $g_k(x) \le 0$, $k = 1, 2, \ldots m$
where $g_0, g_1, \ldots\ldots, g_m$ are polynomial functions.

Note that GP is a convex programming problem whereas GGP is a non-convex programming problem [2]**.** By using the variable substitution $lg(x)=w$, GP can be transformed into a linear programming problem [13]. The non-convex GGP can be transformed into a sequence of convex GP problems using the geometric-arithmetic inequality, and each GP can be solved individually [1]. This method is quite effective.

This paper is organized as follows: the details of our algorithm for solving a single critical path problem are given in section II; extensions to k critical paths are discussed in section III; experimental results are presented in section IV; conclusion and future work are discussed in section V; and references are listed in section VI.

* This work was funded in part by NSF grant No. MIP-9628999, and by SRC contract No. 98-DJ-606.

II. The Proposed Algorithm

Problem Formulation: Given a mapped and placed network, a technology library, and assuming that all input signals arrival at time 0 and all outputs should be ready at time t_{cycle}, the goal is to minimize t_{cycle} by restructuring and re-positioning gates in the circuit.

In this section, we present our algorithm to solve the above problem one critical path at a time. The extensions to *k* critical paths are discussed in section III.

Our proposed algorithm is outlined in Figure 2. After timing analysis, we cluster the circuit by constructing a set of *super-cells* along the critical path. A set of re-mapping solutions is then generated for each super-cell. Each set consists of all non-inferior re-mapping solutions for the super-cell. During solution generation, we use a placement algorithm concurrently with the re-mapping procedure to estimate the wire delays and wire loads. The placement algorithm generates a rough placement which will be refined in the subsequent steps. In this step, no re-mapping solution is selected for any super-cell. In the next step, the best mapping and placement solutions for all super-cells are obtained concurrently by solving an appropriately defined GGP problem. After that, the layout will be updated to reflect the required changes. Throughout these optimization steps, constraints such as timing requirements and repositioning constraints are computed and enforced to ensure timing closure of this iterative optimization algorithm. This algorithm iterates until the design goal is met or there is no more room for further improvement.

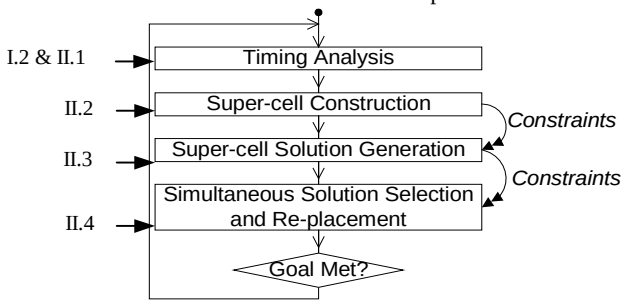

Figure 2. Algorithm Overview.

II.1. Timing Analysis and Slack Distribution

The first step of our algorithm is to perform timing analysis using the model presented in section I.2 to identify the critical paths in the circuit. An example consisting of 9 logic gates is given in Figure 3. The most critical path from input I_5 to output O_4 is marked with thicker lines.

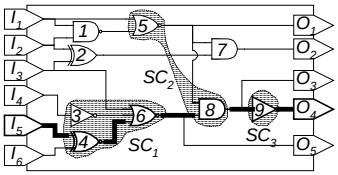

Figure 3. Super-cell Generation.

Based on the timing analysis results, delay slacks are distributed to each gate. We use the *zero-slack-algorithm* of [12]. The main idea is to assign more slacks to the gates that are *slack sensitive*. Please refer to [12] for details of this algorithm.

II.2. Super-cell Construction

In this section, we introduce our clustering algorithm. This algorithm clusters the circuit by constructing a set of super-cells along the critical path. These super-cells are the targets for further optimizations.

Definition 3: A *critical* gate is a gate on the critical path.

Definition 4: A *super-cell* (SC in short) is a set of logically connected gates with at least one critical gate.

Definition 5: A gate G_i is a *level-L fanin* of G_j if there is a directed shortest path from G_i to G_j containing at most L edges.

```
1.  For each critical gate CG_i on the critical path
2.      SC_i = { CG_i }, where CG_i is called the root of SC_i
3.  For each super-cell SC_i in the order of increasing slacks
4.      For each level-L fanin G_j of CG_i
5.          If G_j is not included in any super-cell yet
6.              SC_i += { G_j }
7.  For each single output CG_i that fanouts to CG_j
8.      Merge SC_i into SC_j
```

Lines 1 to 2 initialize the SCs by assigning the critical gate as the *root* to each one of them. This also prevents a critical gate from being included in other super-cells. Lines 3 to 6 extend a super-cell to include the level-*L* fanins for the root, unless that fanin is already included in another super-cell. Because we sort the super-cells by increasing slacks, if there is a competition for a fanin between super-cells, the most critical super-cell will get that fanin. We want more gates to be included in a more critical super-cell because then the potential for optimizing that super-cell increases. Lines 7 to 8 merge the single-output super-cell to the fanout super-cell. By merging the single-output super-cells, we are able to increase the size of the super-cells without increasing the complexity of our algorithm. In Figure 3, since critical gate 4 has only one output, the super-cell whose root is gate 4 is merged with the super-cell whose root is gate 6, generating a larger SC_1 that contains two critical gates.

Note that this clustering algorithm may produce different solutions depending on the order in which super-cells of the same criticality are enumerated in line 3 of the algorithm.

Lemma 1: The *root* of a super-cell is the only gate that (a) is a critical gate, and (b) has fanouts outside the super-cell.

Proof: From the algorithm, we know that only the merge operation may add other critical gates to a super-cell. However, the merge will happen if and only if that critical gate has only one fanout which must be inside the super-cell. Therefore, after the merge, that critical gate no longer fanouts to any other gates outside the super-cell. ■

Corollary 1: Among all the output ports of a super-cell, there is one and only one output port that is on a critical path.

Lemma 2: There is a path from each input port of a super-cell to the critical output port.

Proof: The *root* of the super-cell is driving the critical output port. From the algorithm, we know that each gate in the super-cell is in the transitive fanin cone of the root. Therefore, there must be a path from each gate in the super-cell to the root and the critical output port. ■

II.3. Solution Set Generation for Super-Cells

There exist many re-mapping solutions for a super-cell. Using our delay model, some of them are considered to be inferior and dropped from further consideration, thus greatly reducing the complexity of our algorithm.

II.3.1. Delay modeling

Lemma 3: For a given DAG circuit, when considering only one critical path, at most one input port of any super-cell lies on the critical path.

Proof: Assume that there is more than one input port that is on the same critical path. This means that the critical path has to leave the super-cell from the critical output and return to the super-cell at least once. From Lemma 2, we know there is a path from each critical input port to the root. This creates a forward loop in the circuit and violates the assumption that we have a DAG. ■

Based on Corollary 1 and Lemma 3, we can regard each super-cell as a black box that has only one critical input port and one critical output port as shown in Figure 4. The design parameters that are relevant to delay calculation are the input capacitance C_{in} for the input port, the driver strength R_{out} for the output port, and the internal delay τ of the super-cell. τ is used directly in the delay calculation; C_{in} affects the extrinsic delay of the previous stage; and R_{out} is used to calculate the extrinsic delay of the current super-cell.

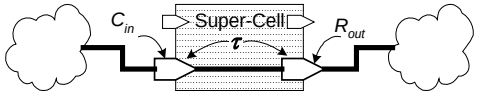

Figure 4. Delay Parameters.

Note that even though we do not include area as a parameter, we indeed consider area/delay tradeoff for the non-critical gates during the construction of the solution set. Details are given in section II.3.2.

For a given mapped solution of a super-cell, R and C can be easily determined from the technology library. R is the driver strength of the root gate, and C is the input capacitance of the critical input port. We define $\boldsymbol{\tau}$ as the total delay from the critical input to the critical output excluding the extrinsic delay of the root. τ includes both the gate delays and the internal wire delays. The inter-super-cell wire delays are added when the super-cells are being placed. In the example shown in Figure 3, τ for SC_1 is the summation of the delay of gate 4, the wire delay between gate 4 and 6, and the intrinsic delay of gate 6.

II.3.2. Dynamic programming

A set of re-mapping solutions will be generated for each super-cell. Each solution contains three values: R, C and τ. We will keep only the non-inferior solutions in the solution set. The inferiority of solutions is defined as follows:

Definition 6: A solution $P_1(R_1,C_1,\tau_1)$ is said to be inferior to another solution $P_2(R_2,C_2,\tau_2)$ if and only if $R_1 \geq R_2$, $C_1 \geq C_2$, and $\tau_1 \geq \tau_2$ are all true.

Theorem 1: Assuming there are two solutions P_1 and P_2 for a super-cell SC, if P_1 is inferior to P_2, then choosing P_2 instead of P_1 for SC cannot increase the delay of the whole critical path.

Proof: The delay of the critical path d_p is $a_{SC}+\tau+R\times c_{load}+d_{SC\rightarrow PO}$, where a_{SC} the arrival time of the critical input port of SC, c_{load} is the output load of the critical output port, and $d_{SC\rightarrow PO}$ is the delay from the critical output port of SC to the primary output of the critical path. The values of c_{load} and $d_{SC\rightarrow PO}$ are constants for SC regardless of the solution being selected. $a_{sc1} \geq a_{sc2}$ because $C_1 \geq C_2$. Combining with the facts that $R_1 \geq R_2$ and $\tau_1 \geq \tau_2$, we conclude that $d_{p1} \geq d_{p2}$. ■

We use a dynamic programming based algorithm to find all non-inferior solutions for every super-cell. We perform simultaneous technology mapping and mincut-based placement similar to that used in [10]. The objective of the linear placement is to minimize the cut density of the placement. We focus on routing congestion inside the super-cells because we believe that most of the interconnections among the gates inside a super-cell are local and thus short. Wire congestion is more important than wire delay for short connections.

During the dynamic programming, we do not need to compute $(\boldsymbol{R,C,\tau})$ values at each internal node. Because R is only related to the critical output, only matches containing the critical output need to compute it. Similarly, because C is only related to the critical input, only matches whose transitive fanin cone contains the critical input need to store it. Moreover, in order to consider area/delay tradeoff for non-critical nodes, the gate area A is added to any match that contains only non-critical nodes. Therefore, we have three different curves during the dynamic programming: a 3-D $<R,C,\tau>$ curve, 2-D $<C,\tau>$ curves, and 2-D $<A,\tau>$ curves. The root node will have a 3-D $<R,C,\tau>$ curve; any critical node will have a 2-D $<C,\boldsymbol{\tau}>$ curve; and other nodes will have the 2-D $<A,\tau>$ curves. For the technology decomposed super-cell shown in Figure 5, root node 3 has a 3-D curve; critical nodes 1 and 2 have 2-D $<C,\tau>$ curves; and non-critical nodes 4 and 5 have 2-D $<A,\tau>$ curves. The inferiority of the solutions on the 2-D curves is defined similar to Definition 6. We can easily prove that inferior points can be safely discarded using arguments similar to Theorem 1.

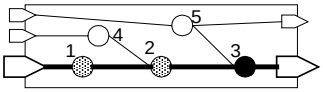

Figure 5. Curves used in Solution Generation.

The pseudo code of solution generation is given as follows:

```
1.  Generate a delay constraint for each non-critical output
2.  Technology decompose the super-cell
3.  For each node n in the reversed depth-first-search order
4.      For every match m rooted at n
5.          Mincut place subtrees rooted at the input of m
6.          If m contains the root node
7.              Generate 3-D <R,C,τ> curve
8.          Else if m contains any other critical node
9.              Generate 2-D <C,τ> curve
10.         Else Generate 2-D <A,τ> curve
11.     Prune the curve for n
12. Generate repositioning constraint Δs for each solution
```

As a first step we generate a delay constraint for each non-critical output port P. These constraints help control the perturbations to other parts of the circuit. It is given as: $a_P \leq \alpha \times req_p$, where α is a constant (greater or equal to one) that controls the degree of delay increase we can tolerate.

Lines 4 to 10 perform simultaneous technology re-mapping and mincut placement. Here we compute the relevant curves for each node. R, C, A values are found from the match, and τ is calculated based on both the match and the placement solution. We explain how to calculate τ using the example shown in Figure 6. In Figure 6, we are generating the solutions for SC_1 in Figure 3. Assume that at some step of the dynamic programming approach, gate 3's 2-D $<A,\tau>$ curve is already computed, and so is gate 4's 2-D $<C,\tau>$ curve. We now generate the 3-D $<R,C,\tau>$ curve for root gate 6 from these curves. Assuming a match m is found, and the match's inputs are gates 3 and 4. The mincut placement has determined that gate 4 should be placed between gates 3 and 6. Note that the linear placement only determines a linear ordering of the gates; hence the absolute coordinates are not known. The estimated wire length is based on this linear order. In this example, the estimated wire length between gates 4 and 6 l_{46} is computed as $\beta \times d_{46}$, where d_{46} is the distance between gates 4 and 6 in the linear placement, and β is constant that is greater than one. τ is computed as $\tau_{gate4}+drive_{gate4}\times \text{load}_{\text{gate4}}+w_{46}$, where $\text{load}_{\text{gate4}}$ the summation of the wire load estimation plus the input capacitance of match m for gate 4, and w_{46} is the estimated wire delay between gates 4 and 6 computed from l_{46}. The solution of gate 3 with the smallest area that satisfies $\tau_{gate3}+drive_{gate3}\times \text{load}+w_{36} \leq \tau$ is chosen thereafter.

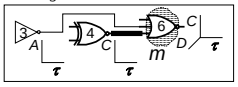

Figure 6. Curve Generation.

Line 11 prunes the curve and line 12 generates the repositioning constraints for each re-mapping solution for the super-cell. The pseudo code is given as follows:

```
1.  For each re-mapping solution s of the super-cell
2.      For non-critical port Pi on the super-cell
3.          Let asi be the arrival time of Pi using s
4.          Δsi=displacement(MIN(req(fanout of Pi)-asi)
5.      Δs = MIN(Δsi)
```

where *displacement* is a function to transform the slack into the allowable wire displacement. This constraint is illustrated in Figure 7. When repositioning SC, the wire delay between a non-critical port P and its fanout may be changed. If the wire delay gets too large, the path between P and its fanout may become the new critical path, which is undesirable. Therefore, this constraint restricts the movement of gate 6 in the subsequent optimization phases so that a new critical path is not created.

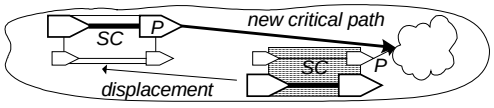

Figure 7. Repositioning Constraints.

Theorem 2: Subject to the errors introduced by the wire estimation process, the 3-D curve for a super-cell includes all non-inferior re-mapping solutions.

Proof: The only solutions that we do not keep are the inferior solutions. From Theorem 1, we know that we can safely discard these inferior solutions. Therefore, our final solution curve includes all non-inferior re-mapping solutions. ■

II.3.3. Complexity analysis

The following analysis assumes that the number of gates in the technology library is a constant, and n denotes the number of nodes in the decomposed super-cell.

Lemma 4: The maximum number of points in the $<A,\tau>$ curve is O(n).

Proof: Normalize the area of the gates in the technology library to be all integers. Assume that the largest gate area in the library is M, so the maximum possible gate area for the super-cell is O(nM). Because M is a constant, we have maximum O(n) distinctive gate area values. Because two solutions will never have the same gate area (one will be eliminated as an inferior solution), the maximum number of solutions in the $<A,\tau>$ curve is O(n). ■

Lemma 5: The maximum number of points in the $<C,\tau>$ curve is O(1).

Proof: Because the number of gates in the library is a constant, the maximum number of distinct C is also a constant. Use the same argument as in the proof of Lemma 4, we know the maximum number of solutions in $<C,\tau>$ curve is a constant. ■

Lemma 6: The maximum number of points in the $<R,C,\tau>$ is O(1).

Proof: Using the same argument as in the proof of Lemma 5, we know that the maximum number of distinct R is a constant. This implies that the maximum number of solutions in $<R,C,\tau>$ curve is a constant. ■

Theorem 3: The runtime complexity of the solution generation step for a super-cell is O($n^2 lgn$).

Proof: The runtime to generate a $<A,\tau>$ curve is given in [4] as O($nlgn$). Similarly, the runtime to generate a $<C,\tau>$ curve and a $<R,C,\tau>$ curve are both O(lgn), which is dominated by the generation of the $<A,\tau>$ curve. Moreover, the curve generation process will be repeated for each node, so the total runtime is O($n^2 lgn$). ■

II.4. Concurrent Solution Selection and Placement

Simply choosing the best solution for each super-cell may actually lead to poorer circuit timing because of the complex dependencies between the super-cells. We select the best mapping and placement solutions for all super-cells at the same time. A naive approach is to define a binary variable for each solution, and use a mixed integer linear programming (MILP) to solve the problem. However, it is impractical to solve the resulting MILP for any reasonably sized circuits due to its high complexity. Here we use a generalized geometric programming (GGP) formulation to address this problem.

II.4.1. Interpolation function

One of the requirements of the GGP is that all variables in the problem formulation be continuous. While it is natural to have continuous variables to model the placement, our solution set for each super-cell is indeed discrete. Therefore, the first step we need to do is to make the solution set continuous.

Intuitively, the intrinsic delay τ for any critical path is related to the size of the gates along the path, that is, $\tau=f_1(Z_1,Z_2,\ldots,Z_n)$, where Z_i is the sizing variable for the i^{th} gate on the critical path as shown in Figure 8. However, because Z_2 to Z_{n-1} are internal to the super-cell, we can simplify the function as $\tau=f_2(Z_1,Z_n)$, where Z_1 is the gate that receives an input to the super-cell, and Z_n is the gate that drives the output signal of the super-cell. Moreover, because C and R are directly related to Z_1 and Z_n respectively, we can express the function as $\tau=f_3(C,R)$.

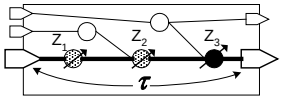

Figure 8. Super-cell modeling.

Even though the exact form of f cannot be found, the portion of f that is within the data range of the solution set can be obtained with sufficient accuracy using Lagrange interpolation [11]. We can write the interpolated function as:

$$\tau = F(C,R) \quad (4)$$

The wire load c_{net} and wire resistance r_{net} are computed based on the placement information using the approach described in section I.3. c_{load} is computed as:

$$c_{load} = \Sigma C_j, \ \forall \text{fanout } j \quad (5)$$

For a super-cell, the delay equation (1) becomes:

$$d_{ij}=\tau_i+R\times(c_{net}+c_{load})+r_{net}\times c_{load} \quad (6)$$

Lemma 7: Function F is a polynomial function.

II.4.2. Repositioning constraints

For any given super-cell, the repositioning constraint Δ_s exists as an additional data for every solution of this super-cell. Every solution will have its own repositioning constraint Δ_s, which is in general different from one solution to next. Therefore, a separate function should be constructed for the repositioning constraint for a super-cell using a similar approach as that of constructing function f. While our algorithm is capable of accepting the repositioning constraint in any polynomial function form, we feel it is too costly and unnecessary to do so. The reason is that the wire delay used in deriving the constraints is calculated based on estimation, which is already inexact. Therefore we have chosen to use a single repositioning constraint $\Delta=MIN(\Delta_s)$ for each super-cell. Note that each super-cell has its own repositioning constraint Δ, and the value differs from one super-cell to next.

II.4.3. Generalized geometric programming

The generalized geometric programming of the simultaneous solution selection and repositioning problem can be stated as follows:

minimize	t_{cycle}	
s.t.	$a_j \geq a_i + d_{ij}$	$\forall$ SCs
	$a_j \leq t_{cycle}$	$\forall$ primary outputs
	$a_j \geq 0$	$\forall$ primary inputs
	$\lvert x_i-x_i' \rvert \leq \Delta_x$	$\forall$ SCs
	$\lvert y_i-y_i' \rvert \leq \Delta_y$	$\forall$ SCs

There are four variables for each super-cell, x, y, R and C. Therefore, we have a total of $4\times N$ variables, where N is the number of super-cells in the circuit.

Theorem 4: The above problem is a GGP problem.

Proof: Follows from Definition 2 and Lemma 7. ■

The GGP solver gives the best values of x, y, R and C for each super-cell. τ can be calculated using (4). It is likely that we are not able to find the exact (R, C, τ) combinations in the original (discrete) solution set. We will then pick the solution with minimal overall percentage error as the solution for that super-cell.

II.4.4. Layout update

After the re-mapping and repositioning solution has been chosen for each super-cell, we need to update the layout to remove overlap and/or congestion. As we have discussed in section II.3, the gates inside the super-cells have only the rough placement available. Therefore, we need to place these local gates. We put them back to the available slots using a linear assignment approach [9]. The available slots are those positions that were occupied by the gates in the original implementation of the super-cells, and those that are defined as *free* by the designers. The cost of assigning a gate G_i inside a super cell SC_j to an empty slot E_k is given by:

$$Cost = Dis(SC_j, E_k)+ \gamma\times AreaMatch(G_i, E_k) \quad (7)$$

where *Dis* denotes the distance between the super-cell and the slot; *AreaMatch* computes the area mismatch between the gate and the slot; and γ is a constant less than one. Due to the space limitation, we will not present the details here.

III. Extensions to Multiple Critical Paths

In this section, we extend our algorithm to process up to *k* critical paths at the same time, where *k* is a user-specified parameter. The larger *k* is, the better the final result, but the slower the runtime.

Theorem 5: If the critical paths do not intersect with each other, our algorithm can process the critical paths independently.

Proof: Figure 9 shows a non-critical gate *G* which is shared by *k'* different critical paths. The critical paths themselves do not intersect. During the re-mapping of *G*'s fanins, we generate the constraints such that *G*'s arrival time will not exceed its required time. Moreover, if *G* included in a super-cell *SC* as shown in the figure, when we construct the solution set for *SC*, we also generate the constraints that the arrival times of *G*'s fanouts do not exceed their required times. Therefore, the solution selection for super-cells on different critical paths can be performed at the same time. ■

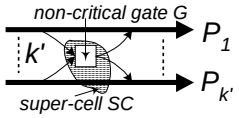

Figure 9. Non-intersecting Critical Paths.

We however have to extend our algorithm when there are *k'* critical paths intersecting at a gate. When we are building the solution set, we need to record the *C* and τ for each critical path. Because we still have one gate driving the critical output, we need only one *R*. Therefore, the final curve generated at the root of any super-cell will have $2k'+1$ dimensions, including *k' C's*, *k' τ's*, and one *R* values, as shown in Figure 10. In the worst case, *k'* will be as large as *k*, which is the number of critical paths that we are considering at the same time. However, it is rare to see that all critical paths use the same gate. Actually, the number of paths sharing the same gate is rather small in our benchmarks, thus greatly reducing the complexity of our algorithm.

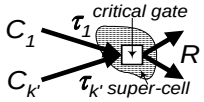

Figure 10. Intersecting Critical Paths.

Moreover, because there are $2k'+1$ values, we need an interpolation function with $2k'$ parameters. Considering the fact that we have only one *R*, we can rewrite the interpolation function as:

$$R=F(\tau_1,\tau_2,\ldots,\tau_{k'},C_1,C_2,\ldots C_{k'}) \quad (8)$$

We keep the same generalized geometric programming formulation except to use the above equation for computing delay through super-cells. Moreover, the number of variables is increased to $\sum_{\forall i} 2k_i + 2$, where k_i is the number of critical paths for the i^{th} super-cell.

IV. Experimental Results

We have implemented our algorithm PRTC (Placement and Re-mapping for Timing Closure) in C++. The input circuits to PRTC are technology independently optimized, mapped, placed and globally routed using timing-driven algorithms. We then apply PRTC to improve their timings. We use an industrial strength 0.35µ ASIC library to generate the results. The runtimes of C1355 and C3540 are 55 seconds and 125 seconds, respectively. These runtimes are obtained by running PRTC on a Sun Ultra-Sparc workstation with 256MB memory. The experimental results for all of the recommended benchmarks in IWLS95 are presented in Table 2. The first column gives the name of the benchmarks. The second and third columns give the number of gates and nets in the original circuits. The fourth and fifth columns are the total area and total delay for the original circuit, and the last two columns give the ratio of total area and total delay after running PRTC. On average, we are able to improve the postlayout timing by 29%, while keeping the area increase to 5%.

			Original		PRTC	
	Gates	Nets	Area	Delay	Area	Delay
C1355	324	367	2637105	7.15	1.06	0.88
C1908	528	563	4852287	12.04	1.04	0.72
C2670	464	699	4755861	13.15	1.14	0.62
C3540	840	892	9026112	21.98	1.05	0.71
C432	239	277	2142138	8.71	1.05	0.89
C6288	2377	2411	19086244	43.35	0.98	0.68
C7552	1304	1513	12817955	13.08	0.99	0.65
b9	83	126	586430	2.87	1.07	0.76
dalu	474	551	5262660	17.42	1.03	0.53
des	1741	1999	24550920	20.08	1.09	0.71
k2	700	747	8558596	17.17	0.96	0.70
rot	494	631	4635405	8.97	1.14	0.92
t481	351	369	3536687	12.05	1.11	0.46
					1.05	0.71

Table 2. Experimental Results.

V. Conclusions

In this paper, we presented a new algorithm that simultaneously performs logic restructuring and placement on a circuit. This algorithm constructs super-cells along the *k* most-critical paths and then generates all non-inferior re-mapping solutions for the critical part of the circuit. Finally, it selects the best mapping solution for all super-cells while repositioning them concurrently. Our future work is to include more optimization steps such as fanout optimization techniques into this algorithm.

VI. Reference

[1] M. Avriel, R. Dembo, U. Passy, "Solution of Generalized Geometric Programming", in *International Journals for Numerical Methods in Engineering*, vol.9, 1975

[2] C. Beightler, D. T. Philips, "Applied Geometric Programming", 1976

[3] M. Berkelaar, "Area-Power-Delay Trade-off in Logic Synthesis", *Ph.D Thesis*, Eindhoven University of Technology, 1992

[4] K. Chaudhary, M. Pedram, "Computing the Area versus Delay Trade-off Curves in Technology Mapping", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 12, pp.1480-1489, 1995.

[5] W. Chen, C-T. Hsieh, M. Pedram, "Gate Sizing with Controlled Displacement", in *Proceedings of International Symposium on Physical Design*, pp.127-132, 1999.

[6] F.R.K.Chung, F.K. Hwang, "The Largest Minimal Rectilinear Steiner Trees for a Set of N Points Enclosed in a Rectangle with Given Perimeter", *Network*, 9:19-36, 1979

[7] O. Coudert, R. Haddad, "New Algorithms for Gate Sizing: a Comparative Study", in *Proceedings of 33rd Design Automation Conference*, pp.734-739, Jun 1996

[8] J.P. Fishburn, A.E. Dunlop, "TILOS: a Posynomial Programming Approach to Transistor Sizing", in *Proceedings of International Conference on Computer Aided Design*, pp.326-328, 1985

[9] T. Lengauer, "Combinatorial Algorithms for Integrated Circuit Layout", *John Wiley & Sons Ltd.*, 1990

[10] J. Lou, A. H. Salek, and M. Pedram, "An Exact Solution to Simultaneous Technology Mapping and Linear Placement Problem", in *Proceedings of International Conference on Computer Aided Design*, pp.671-675, 1997.

[11] J. Morris, "Computational Methods in Elementary Numerical Analysis", *John Wiley & Sons Ltd.*, 1983

[12] R. Nair, C.L. Berman, P.S. Hauge, E.J. Yoffa, "Generation of Performance Constraints for Layout", in *IEEE Transaction of Computer-Aided Design*, pp.860-874, CAD-8(8), 1989

[13] K.O. Kortanek, X. Xu, Y. Ye, "An infeasible interior-point algorithm for solving primal and dual geometric programs", *Mathematical Programming 76*, pp.155-181, 1996