Retiming-Based Timing Analysis with an Application to Mincut-Based Global Placement

Jason Cong and Sung Kyu Lim

***Abstract*—In this paper, we formulate the physical planning with retiming problem and propose an algorithm called GEO. Our performance-driven global placement algorithm GEO is mincut-based, where a multilevel partitioning is performed recursively to divide the netlist and assign gates to the tiles in a top-down fashion. The contribution of our work is on the development of retiming-aware timing analysis (RTA) that is used to guide our mincut-based global placement. Compared to the conventional static timing analysis, RTA provides timing slack information after retiming so that the clock period after retiming can be directly minimized during the placement. We show how to make an effective use of RTA timing slack information in a multilevel partitioning framework. Simultaneous consideration of partitioning and retiming under the geometric delay model enables GEO to hide global interconnect latency more effectively compared to the conventional approaches. In our comparison to the state-of-the art methods that perform partitioning, retiming, and simulated annealing-based floorplanning separately, GEO obtains significant improvement on retimed delay, while maintaining comparable wirelength and runtime results.**

***Index Terms*—Global placement, retiming-based timing analysis.**

I. Introduction

As we move from 0.18-μm technology with a clock frequency of 1–2 GHz to the 0.07-μm technology with a clock frequency of 4–5 GHz in a few years, interconnect delay will continue to far exceed device delay and become a dominating factor in determining the system performance. The majority of the clock period is no longer spent on computing or generating the data but on transmitting and communicating the data between various parts of the chip. Recent advances on interconnect optimization techniques, such as interconnect topology optimization, optimal buffer insertion and sizing, optimal wire-sizing, etc., can help to reduce interconnect delays significantly [1]. However, they are not able to reverse the trend of the growing gap between device and interconnect performance. Therefore, the interconnect-centric paradigm is needed at every level of the design process, from system-architecture design to the careful process of geometry optimization. to eliminate the interconnect performance bottleneck.

Global placement has drawn significant attention from the computer-aided design community in recent years as an indispensable way to handle the complexity of the modern placement problems. The placement problem is divided into global and detailed placement in much the same way as in the division of global and detailed routing. Global placement determines the region for a group of cells to be located whereas detailed placement removes overlap and performs legalization in each region while preserving the global placement solution as much as possible. Typically the two-dimensional (2-D) placement region is divided into $m \times n$ tiles, and cells are placed at the center of the tiles during the global placement. There are three major approaches in global placement—mincut [2], [3], analytical [4], [5], and simulated annealing (SA)-based [6] approaches. In the mincut-based approach, recursive partitioning is performed to divide the netlist and assign gates to the tiles in a top-down fashion. Due to the fast runtime and flexibility in handling various constraints, the mincut-based approach has been used in many modern state-of-the-art placement algorithms [7]–[9]. Global placement has a tremendous impact on the performance since it determines the delay of global connections. Thus, an effective performance-driven partitioning algorithm is crucial in developing mincut-based global placement for performance optimization.

Circuit partitioning divides a given circuit into a collection of smaller subcircuits while satisfying the given area and/or pin constraints. The conventional objective of partitioning is to minimize the number of connections among the subcircuits, which has a direct impact on the final chip area minimization. Under the new interconnect-centric design paradigm, however, partitioning is seen as the crucial step that defines the local and global interconnects [10]—intrapartition connections become local interconnects, whereas interpartition connections become global interconnects. To meet the performance requirement of today's complex design, partitioners must consider the amount of interconnect induced by partitioning (measured by its cutsize) as well as its impact on performance (measured by its delay). Cutsize minimization helps to lower the possibility of critical paths crossing partition boundary multiple times, thus improving performance. However, cutsize minimization alone is not enough since we need more rigorous approaches that address the impact of partitioning on delay minimization.

Performance-driven partitioning methods can be grouped into ones that are designed for combinational circuits [11]–[13], and for sequential circuits with retiming [14]–[16]. Retiming [17] is a sequential logic-optimization technique that exploits the flexibility provided by repositioning flip-flops (FFs) to minimize either the delay or the number of FFs in the circuit. Recently, retiming has become more attractive in handling global interconnects; it allows multiple clock cycles to propagate signals across the chip. Traditionally, retiming is performed during logic synthesis to minimize the longest path delay. However, this step ignores routing delay and, thus, considers only the gate delay. Retiming can be performed after partitioning as a postprocess based on a rough-routing delay estimation. Recent advances in combining partitioning with retiming [14]–[16] enable the partitioner to explore wider solution space that considers equivalent FF movement. However, all of these existing works are based on a very simple "variable gate delay model" [12]—variable delay values for gates, but with a single constant value for interblock connection. This limitation is simply due to the fact that the location and dimension of blocks are not available during the partitioning. Therefore, several recent studies on layout-driven retiming [18], [19] attempted to tight-couple retiming and placement together.

Given a circuit consisting of both fixed or flexible blocks and a netlist interconnecting these blocks, floorplanning constructs a layout by determining the position and shape of each block such that all nets can be routed and the total layout area is minimized. Traditionally, partitioning is performed on top of a retimed circuit to generate blocks, followed by the subsequent floorplanning to determine the location of blocks. However, the separation among partitioning, retiming, and floorplanning poses three major shortcomings for performance optimization: 1) retiming and partitioning suffer from nonrealistic delay estimation, which in turn may mislead performance optimization as illustrated in Fig. 1(a)–(b); 2) a restriction exists in retiming global interconnects—we cannot place FFs along wires but only at the beginning of the wires as illustrated in Fig. 1(c)–(e); and 3) the

Manuscript received July 12, 2003; revised January 5, 2004. This work was supported in part by the MARCO/DARPA Gigascale Silicon Research Center, in part by a grant from the Intel Corporation, in part by the National Science Foundation/Georgia Tech. Packaging Research Center, and in part by Georgia Yamacraw. This paper was recommended by Associate Editor S. Hassoun.

J. Cong is with the Department of Computer Science, University of California, Los Angeles, CA 90095 USA.

S. K. Lim is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: limsk@ece.gatech.edu).

Digital Object Identifier 10.1109/TCAD.2004.837718

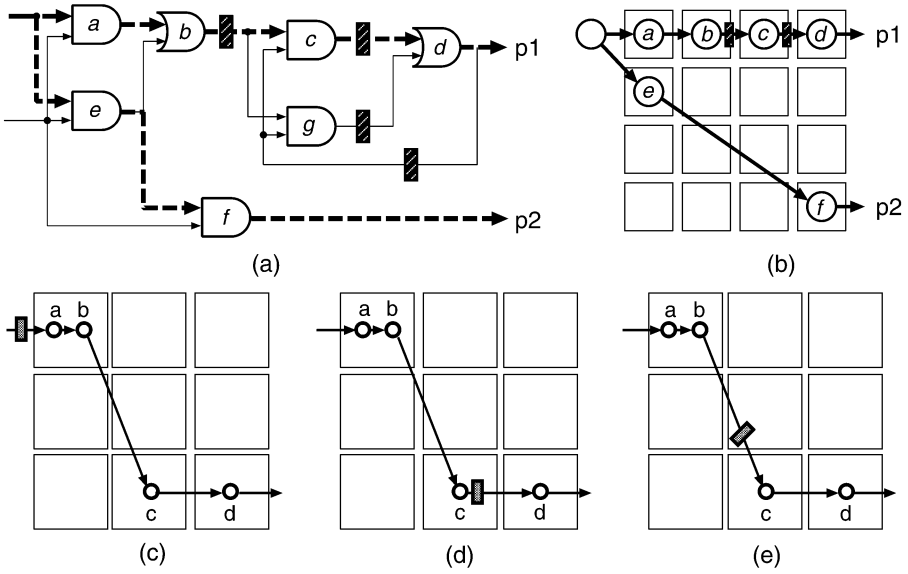

Fig. 1. Importance of geometric delay model. (a)–(b) Let path $p_1 : a \rightarrow d$ and $p_2 : e \rightarrow f$. Assume delay of the gates is d, and interblock edge delay is D. A sequential circuit after retiming and after floorplanning are respectively shown in (a) and (b). Let $d(p)$ denote the delay along a path p. Then, $d(p_1) = d(p_2) = 2d + D$ under the conventional variable gate delay model. However, $d(p_1) \ll d(p_2)$ based on the geometric embedding. (c)–(e) nongeometric embedding based retiming of p_1 gives solution (d) p_1, while finer-grained FF placement based on geometric embedding gives solution (e). Note that the potential improvement can be as big as the delay of the global interconnect itself.

subsequent floorplanning may be constrained by the predefined local and global interconnects from the prior partitioning.

Our performance-driven global placement algorithm GEO is mincut-based, where a multilevel partitioning is performed recursively to divide the netlist and assign gates to the tiles in a top-down fashion. The contribution of our work is on the development of retiming-aware timing analysis (RTA) that is used to guide our mincut-based global placement. Compared to the conventional static timing analysis (STA), RTA provides timing-slack information *after* retiming so that the clock period after retiming can be directly minimized during the placement. We show how to make an effective use of RTA timing-slack information in a multilevel partitioning framework. We also show that the mincut placement environment enables GEO to exploit the more realistic geometric delay model. Simultaneous consideration of partitioning and retiming under the geometric delay model enables GEO to hide global interconnect latency more effectively compared to the conventional approaches [14]–[16]. In our comparison to the state-of-the art methods that perform partitioning, retiming, and SA-based floorplanning separately, GEO obtains significant improvement on retimed delay while maintaining comparable wirelength and runtime results.

The remainder of the paper is organized as follows. Section II provides the problem formulation. Section III presents RTA algorithm. Section IV presents GEO algorithm. Section V provides experimental results. Section VI concludes the paper with the discussion of our ongoing research.

II. Problem Formulation

Given a sequential gate-level netlist NL(C, N), let C denote cells consisting of gates and FFs, and N denote nets that connect the cells. The purpose of the physical planning with retiming (PPR) problem is to assign cells in NL to a given $m \times n$ slots while satisfying the prescribed area constraints. Given a PPR solution P, our primary objective is to minimize *retiming delay* induced by P (to be defined soon).[1] As secondary objectives, we minimize *wirelength* induced by P. The formal definition of the problem is as follows.

Definition 1 (PPR Problem): The PPR problem with the given netlist NL(C, N), a set of $m \times n (= K)$ slots $S = \{S_1, S_2, \ldots, S_K\}$, with $S_i \subset C$, and area constraints $A = (L_i, U_i)$ for $1 \leq i \leq K$ has a solution $P : C \rightarrow S$, where each cell in C is assigned to a unique slot. P is optimal if it satisfies the following conditions: 1) $L_i \leq |S_i| \leq U_i$ for $1 \leq i \leq K$; 2) $S_1 \cup S_2 \cup \cdots \cup S_K = C$; 3) $S_i \cap S_j = \emptyset$ for all $i \neq j$; and 4) retiming delay is minimized.

A. Geometric Delay Model

The delay model in the context of mincut-based global placement specifies how the cell and edge delay values are to be determined. There are two proposed delay models in the literature that many of the existing partitioning works are based on, such as the unit delay model [11] and the variable gate delay model [12] (also called "general delay model" in the literature). In the unit delay model, cell delay is ignored, and the edges that connect vertices in different blocks (= interblock edges) are given a constant delay of D, while intrablock edges incur no delay. In the variable gate delay model, cells can be assigned with arbitrary delay values, and interblock edges are given a constant delay of D, while intrablock edges incur no delay.[2]

Note that both delay models are nongeometric in a sense that we do not consider the actual length of the edges into the delay calculation simply because the cell location is not known during partitioning. However, if we use partitioning to do placement (= mincut placement), we can exploit cell-location information available during top-down placement for more accurate edge delay calculation. In the beginning, all cells are located at the center of the placement region. After the first vertical cut, there are two different cell locations: left and right. After the two subsequent horizontal cuts, there are four different cell locations: upper left, upper right, lower left, and lower right. Therefore, the cell location is getting more refined as we perform more bipartitioning. Therefore, the edge-delay calculation

[1]Note that some literature uses a term "clock period after retiming" to refer to the delay of the longest combinational path after retiming.

[2]Local interconnect delay can be estimated and its average can be lumped into the cell delay $d(v)$ for simplicity.

can exploit the cell location readily available during this mincut-based placement.

In our *geometric delay model*, we first model NL with a directed, edge-weighted graph $G = (V, E)$[3], where the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents cells, and the directed edge set $E = \{e_1, e_2, \ldots, e_m\}$ represents signal directions in NL. A directed edge $e(u, v)$ denotes the connection from vertex u to vertex v. Each vertex v has a delay of $d(v)$, and each edge $e(u, v)$ has a delay of $d(e)$ that is linearly proportional to the Manhattan distance between u and v, i.e.,

$$d(e) \propto |x_i - x_j| + |y_i - y_j|$$

where $u \in S_i$ and $v \in S_j$. The delay of a path $p(u, v)$ from u to v, denoted $d(p)$, is defined to be the sum of $d(e)$ and $d(v)$ along p.

Note that we do not use the nonlinear Elmore delay model since routing topology (= *RC* tree) is not available during the partitioning and global placement stage. A recent study [20] shows that wire delay becomes proportional to its linear length rather than squared length if interconnect optimization schemes, such as wire sizing, buffer sizing/insertion, etc., are followed. We attempted to use the squared manhattan distance for $d(e)$ instead of linear distance and observed little difference in terms of final delay results. We conjecture that the minimization of both linear and squared wirelengths have the similar impact on our mincut-based global placement framework.

B. Static Delay Versus Retiming Delay

In this paper, we consider two kinds of delay values from a global placement solution: *static delay* and *retiming delay*. The static delay refers to the delay of the longest combinational path *before* retiming. The retiming delay refers to the delay of the longest combinational path *after* retiming. The formal definition of static delay is as follows.

Definition 2 (Static Delay): The static delay induced by a PPR solution is the largest delay among all combinational paths from G, where G is the directed graph representation of the given netlist (discussed in Section II-A).

We use the retiming graph [17] for the computation of retiming delay. A retiming graph $R = (V, E, W)$ consists of a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ that represents gates, a directed edge set $E = \{e_1, e_2, \ldots, e_m\}$ that represents signal directions in NL, and edge weight set W that represents the the number of FFs between the two end-vertices of each edge. A retiming is a labeling of the vertices $r : V \rightarrow Z$, where Z is the set of integers. The weight of an edge $e = (u, v)$ after retiming is denoted by $w^r(e)$ and is given by $w(e(u, v)) + r(v) - r(u)$. The retiming label $r(v)$ for a vertex $v \in V$ represents the number of FFs moved from its output toward its inputs. A circuit is retimed to a delay ϕ by a retiming r if the following conditions are satisfied; 1) $w^r(e) \geq 0$ and 2) $w^r(p) \geq 1$ for each path p such that $d(p) > \phi$, where $w^r(p) = \sum_{e \in p} w^r(e)$. Since ϕ after retiming is usually smaller, retiming is used to further reduce the delay.

For a given PPR solution P and a *target delay* ϕ, the edge length of $e = (u, v)$, denoted $l(e)$, is defined to be $-\phi \cdot w(e) + d(v) + d(e)$. The path length of p, denoted $l(p)$, is $\sum_{e \in p} l(e)$. The sequential arrival time (SAT) [14] of v, denoted $l(v)$, is the maximum path length from primary inputs (PIs) to v. If the SAT for all primary outputs (POs) are less than or equal to ϕ, the target delay ϕ is called feasible. Let $D_g = \max\{d(v) \mid v \in V\}$. The formal definition of retiming delay is as follows.[4]

Definition 3 (Retiming Delay): The retiming delay induced by a PPR solution is defined to be the $\phi + D_g$, where ϕ is the minimum feasible target delay from R and R is the retiming graph representation of the given netlist.

C. Wirelength Objective

For the wirelength calculation, we model NL with a hypergraph $H = (V, E_H)$, where the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents cells, and the hyperedge set $E_H = \{h_1, h_2, \ldots, h_{m'}\}$ represents nets in NL. Each hyperedge h is a nonempty subset of V. The x-span of a hyperedge h, denoted h_x, is defined as $h_x = \max_{c \in h}\{x_i \mid c \in B_i\} - \min_{c \in h}\{x_i \mid c \in B_i\}$. The y-span of h, denoted h_y, is similarly defined using y-coordinates instead. The sum of x-span and y-span of each hyperedge h is the half-perimeter of the bounding box (HPBB) of cells in h. The wirelength induced by a global placement solution is the sum of HPBB of all hyperedges.

III. Retiming-Based Timing Analysis

This section presents our RTA engine. We adopt the concept of SAT, which was first introduced in [14] and later on used in [15] and [16] for partitioning with retiming. We extend SAT and introduce *sequential required time* and *sequential slack*. The slack information is used to derive ϵ-network that identifies timing critical cells in the circuit after retiming. In GEO, the derivation of ϵ-network plays a key role in determining the cell move gain.

A. Static Timing Analysis

The concepts of *arrival time, required time*, and *timing slack* are fundamental in the STA of sequential circuits. We perform STA on G, the directed graph representation of the netlist discussed in Section II-A. The arrival time of a vertex $v \in V$, denoted $l_c(v)$, is defined to be the maximum combinational path delay from PIs to v. The computation of arrival time can be done by examining fanin vertices

$$l_c(v) = \max\{l_c(u) + d(e) + d(v) \mid e(u, v) \in E\}.$$

The $d(e)$ and $d(v)$ denote the edge and cell delay, respectively. We can compute the l_c of all vertices by visiting them in a topological order and updating $l_c(v)$. We can assign the required time of a vertex $v \in V$, denoted $q_c(v)$, to specify the timing constraint for v, i.e., the delay from PIs to v is "required" to be $q_c(v)$. After assigning $q_c(v)$, we can compute the required time of all upstream vertices, i.e., all vertices reachable from PIs before arriving at v, by examining fanout vertices

$$q_c(v) = \min\{q_c(u) - d(e) - d(v) \mid e(v, u) \in E\}.$$

We can compute the required time of all vertices by visiting them in a reverse topological order and updating $q_c(v)$. The timing slack of a vertex $v \in V$, denoted $s_c(v)$, is defined to be the difference between the required time and arrival time of v

$$s_c(v) = q_c(v) - l_c(v).$$

The slack is used to determine the timing criticality of vertices in V: a smaller $s(v)$ indicates higher timing criticality for v. We can derive the

[3]In this paper, we use three different models to represent the sequential circuits: 1) G, the directed graph model for STA; 2) R, the retiming graph model (to be discussed in Section II-B) for RTA; and 3) H, the hypergraph model (to be discussed in Section II-C) for computing wirelength.

[4]For the conventional nongeometric embedding-based partitioning with $d(e) = D$ for all interblock edges, the authors of [14] showed that P can be retimed to a delay less than $\phi + D$, if ϕ is feasible. A straightforward extension of this delay bound $\phi + D$ for our PPR problem is $\phi + D_e$, where $D_e = \max\{d(e) \mid v \in E\}$. However, our FF placement phase in Section IV-D reduces this bound to $\phi + D_g$, where $D_g \ll D_e$.

ϵ-network from slack values. ϵ-network is defined to be a subgraph of G consisting of vertices whose slack is smaller than or equal to ϵ.

Definition 4 (ϵ-Network): The ϵ-network $G' = (V', E')$ is a subgraph of G, where $V' = \{v \mid v \in V, s(v) \leq \epsilon\}$, and $E' = \{e \mid e(u, v) \in E, u \in V', v \in V'\}$.

Thus, ϵ-network consists of timing critical vertices that deserve attention for delay optimization.

B. RTA Algorithm

The concepts of SAT, sequential required time, and sequential timing slack are fundamental in RTA of sequential circuits. We perform RTA on R, the retiming graph representation of the netlist discussed in Section II-B. In order to handle sequential circuits directly—as opposed to the conventional approach where FFs are removed to make the circuit combinational—we define the SAT of $v \in V$ as follows:

$$l(v) = \max\{l(u) - \phi \cdot w(e) + d(e) + d(v) \mid e(u, v) \in E\}$$

where $w(e)$ in R denotes the number of FFs along the edge e. Retiming involves a feasibility test of given target delay, and ϕ denotes such a target delay. In a similar way, we define the sequential required time (SRT) of $v \in V$ as follows:

$$q(v) = \min\{q(u) + \phi \cdot w(e) - d(e) - d(v) \mid e(v, u) \in E\}.$$

The sequential slack of v, denoted $s(v)$, is given by $q(v) - l(v)$. Note that R may contain loops with positive weights, depending upon target delay ϕ. Therefore, RTA essentially performs the single-source longest-path algorithm as in the Bellmen-Ford algorithm [21]. In order to compute the minimum feasible target delay, we perform binary search using RTA as a feasibility test.

SAT and retiming are closely related. In fact, the computation of SAT and retiming can be performed at the same time. Consider a path p that starts from a PI u and ends at vertex v. If we want to retime p to satisfy the time constraint ϕ, there must be at least $\lceil l(p)/\phi \rceil - 1$ FFs on p. Since there exists $w(p)$ FFs on p, we can set the retiming value $r(v)$ as $\lceil l(p)/\phi \rceil - 1 - w(p)$. Thus, $r(v) = \lceil l(p)/\phi \rceil - 1 - w(p)$. After rewriting, we get $r(v) = \lceil l(v)/\phi \rceil - 1$.

Our RTA uses a feasible target delay ϕ to compute SAT, SRT, and retiming, all at the same time. RTA algorithm determines if the target delay ϕ is feasible. If so, RTA returns SAT, SRT, and retiming values of all vertices in R. In RTA, SAT for all PIs are set to zero while all others are set to $-\infty$. SRT for all POs are set to ϕ while all others are set to ∞. Then, we can iteratively update SAT and SRT until they converge to their maximum and minimum values, respectively. From these SAT and SRT values, we can compute timing slack and its corresponding ϵ-network. We can also obtain retiming values for all vertices as a byproduct of this step as discussed earlier.

Fig. 2 shows the description of our RTA algorithm. The initialization for SAT $l(v)$, sequential required time $q(v)$, retiming $r(v)$, and predecessor $\pi(v)$ for each vertex is done (lines 1–6). The Boolean flag `done` is used to check if update of any vertex is done (line 8)—if not, we conclude that the target delay ϕ is feasible (line 23). We visit each vertex once in each iteration (line 9), and temporal $l(v)$ and $q(v)$ are computed from examining v's fanin (line 10) and fanout (line 11) vertices. We also remember the fanin vertex that causes the update of $l(v)$ in order to keep track of the longest path to v (line 12). If we find that the SAT of any PO is larger than the target delay ϕ, we conclude that ϕ is not feasible (lines 13–14). Otherwise, if newly computed $l(v)$ is larger than the current value, we update $l(v)$, $r(v)$, and $\pi(v)$ (lines 15–18) and conclude that we need additional iteration (line 19). Also, we update $q(v)$ and conclude that we need additional iteration if newly computed

RTA(R, ϕ)

Input: $R(V, E, W)$ and target delay ϕ
Output: $l(v)$, $q(v)$, $r(v)$, and $\pi(v)$ for all $v \in V$

```
1.  for each vertex v ∈ V
2.      l(v) = −∞;
3.      q(v) = ∞;
4.      r(v) = 0;
5.      π(v) = NULL;
6.  l(v_s) = 0, q(v_t) = φ;
7.  for (i = 1 to |V|)
8.      done = TRUE;
9.      for each vertex v_j ∈ V
10.         t_a = max_{e(u,v_j)}{l(u) − φ · w(e) + d(e) + d(v_j)};
11.         t_r = min_{e(v_j,u)}{q(u) + φ · w(e) − d(e) − d(v_j)};
12.         par = u ∈ FI(v_j) that gives t_a;
13.         if (v_j = v_t and t_a > φ)
14.             return FALSE;
15.         if (t_a > l(v_j))
16.             l(v_j) = t_a;
17.             r(v_j) = ⌈l(v_j)/φ⌉ − 1;
18.             π(v_j) = par;
19.             done = FALSE;
20.         if (t_r < q(v_j))
21.             q(v_j) = t_r;
22.             done = FALSE;
23.     if (done = TRUE)
24.         return TRUE;
25. return FALSE;
```

Fig. 2. Description of RTA algorithm that computes the SAT $l(v)$, sequential required time $q(v)$, retiming $r(v)$, and predecessor $\pi(v)$ for all $v \in V$. RTA also determines the feasibility of the target delay ϕ.

$q(v)$ is smaller than the current value (lines 20–22). If RTA does not converge after $O(V)$ iterations, we conclude ϕ is not feasible (line 25).

The complexity of the Bellman–Ford-type longest-path algorithm is $O(n^2)$ in the worst case since it requires $O(n)$ number of relaxation until all values converge to their maximum. If the circuit contains no positive loops, RTA needs only one iteration of relaxation if the vertices are relaxed in topological order starting with PIs. For a circuit with positive loops, however, a topological ordering is not defined. We observe from the related experiments that the number of iterations is usually small compared to $|V|$. Thus, the complexity of RTA in practice is $O(n)$. Note that the retiming graph R has multiple sources—all PIs. In order to perform the single-source longest-path algorithm on R, we add two special vertices v_s and v_t to V and call them the *source* and *sink* vertices. We add directed edges from v_s to all PIs, and from all POs to v_t. The delay of v_s, v_t, and the edges incident to them are set to zero.

IV. GEO Algorithm

A. Overview of GEO Algorithm

GEO is a mincut-based global placement algorithm, where a multilevel partitioning is performed recursively to divide the netlist and assign gates to the $m \times n$ slots in a top-down fashion. GEO consists of two phases, namely, the construction and FF placement phases. During the construction phase, mincut-based global placement is performed. Our RTA identifies the timing critical nodes in the current subnetlist to be partitioned. Then, GEO computes the weights of the nets in the ϵ-network in such a way that the nets that contain cells with smaller sequential timing slack get higher weights (formula is given in Section IV.C). Lastly, GEO performs multilevel bipartitioning to minimize the weighted cutsize. The horizontal and vertical cuts are alternating in breadth-first manner in GEO so that the first horizontal cut divides the chip area into top and bottom block, and the second vertical cuts further divide the chip area into top-left, top-right, bottom-left, and bottom-right block, etc. During the FF-placement

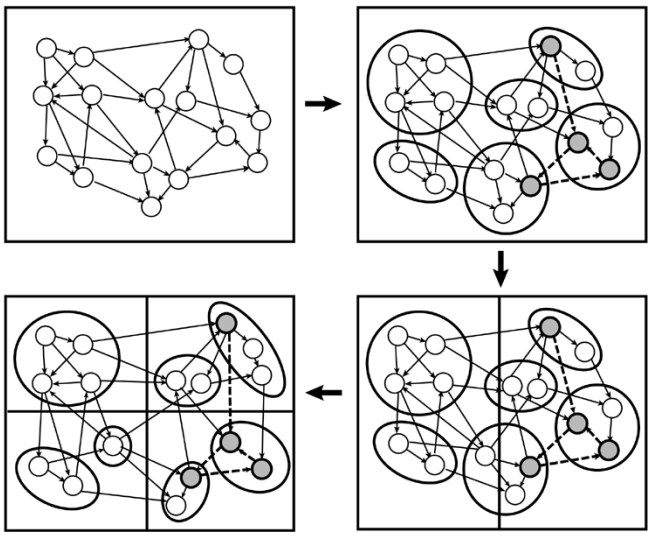

Fig. 3. Illustration of GEO algorithm. We perform a bottom-up multilevel clustering on the input sequential circuit (modeled with a directed graph). Our RTA (Retiming based Timing Analysis) identifies timing critical paths considering retiming (shown in dotted line) and guides the top-down multilevel bipartitioning. The entire placement region is recursively bipartitioned until we obtain the desired number of subregions. Clustering is updated after each bipartitioning.

phase, both coarse-grained and finer-grained repositioning of FFs are performed. The coarse-grained FF repositioning determines which edge gets FFs, and this is done via standard retiming. The finer-grained FF positioning determines at what point of the edge FF is to be located, and this is done with FF placement. An illustration is shown in Fig. 3.

We use iterative improvement-based FM algorithm [22] for partitioning and connectivity-based ESC algorithm [23] for multilevel clustering. We first compute a random bipartitioning among the clusters at the top level and set edge delay in R accordingly. After performing RTA on R and computing weights of nets in the ϵ-network, we perform cell move to reduce the current weighted cutsize. After a pass of cell move is finished, we perform RTA again based on the new solution and update net weights accordingly. This cell move pass continues until we find no gain during the pass. Then this top-level partitioning information is projected onto the next cluster hierarchy level, and more cell move pass is performed on the next level for further refinement. This top-down multilevel partitioning continues until we obtain the partitioning solution at the bottom level. Since RTA provides timing-slack information after retiming, retiming delay is directly minimized during the placement. In addition, the mincut placement environment enables GEO to exploit geometric delay model discussed Section II-A.

B. Construction Phase

A pseudocode of the GEO algorithm is shown in Fig. 4. The inputs to GEO are the netlist NL(C, N), a set S with $m \times n (=K)$ slots, and area bound A. The outputs of GEO are slot assignment $P : C \rightarrow S$ and minimum feasible retiming delay ϕ. GEO consists of two subroutines: GEO-2way recursively bipartitions the netlist, whereas GEO-Kway refines the global placement result. GEO-2way is performed on the subnetlist, whereas GEO-Kway is performed on the entire netlist. A binary tree T is used to keep track of the recursive bipartitioning results—we grow T by adding two children nodes to the current parent node based on the bipartitioning result. Initially, the partitioning tree T has only root node R, and all cells in NL are inserted into R. The first-in first-out (FIFO) queue Q is used to support the recursive breadth-first cut sequence.

```
GEO(NL, S)
Input: netlist NL(C, N), a set S with m × n (= K) slots
Output: slot assignment P: C → S, and retiming delay φ
insert all cells in NL to root node R in T
insert R into Q (= FIFO queue)
while (leaf nodes in T < K)
    N = remove front element in Q
    GEO-2way(N) (= bipartitioning on N)
    split cells in N into N1 and N2
    insert N1 and N2 into Q and T
    if (there are 2^j leaf nodes in T for j > 1)
        GEO-Kway(T)
FF Placement(T)
return P and φ

GEO-2way(N)
NL' = sub-netlist containing cells in N
ESC(NL') (= multi-level clustering on NL')
h = height of the cluster hierarchy
B = random partitioning among clusters at level h
for (i = h downto 0)
    NL'(i) = coarsened NL' at level i
    while (gain)
        DELAY-WEIGHT(NL'(i))
        while (gain)
            minimize weighted cutsize in NL'(i)
            retrieve max gain moves and update B
    project B to level i − 1
return B

GEO-Kway(T)
B = initial partitioning for NL from leaf nodes in T
ESC' (NL) (= preserving K-way cutlines)
perform partitioning to minimize weighted wirelength
update T
```

Fig. 4. Overview of the GEO algorithm.

GEO-2way first generates the subnetlist from the given partition tree node and performs multilevel clustering on it. Then, we obtain a random initial partitioning B among the clusters at the top level of the hierarchy. The subsequent top-down multilevel refinement is used to improve B in terms of retiming delay. We perform RTA to identify timing critical cells and compute the delay weight for the nets in the ϵ network. The subsequent iterative improvement through cluster move tries to minimize the weighted cutsize. Finally, we project the current solution to the next level coarser netlist for multilevel optimization. At the end of GEO-2way, two new children nodes are inserted into T based on B.

GEO-Kway refinement is performed when we obtain 2^j partitions $(j > 1)$ from GEO-2way (4, 8, 16 partitions, etc.). We first perform a restricted multilevel clustering, where grouping among cells in different partitions is prohibited. This allows the partitioner to preserve the initial partitioning results. Then, we perform multilevel partitioning in the same way as in GEO-2way for additional delay improvement. GEO-Kway is applied onto the global netlist for more global level optimization. We use the pairwise movement approach proposed in [24] for our K-way partitioning refinement.

C. Delay-Weight Computation

In GEO, we use sequential slack to compute how much timing slack remains for a gate before it violates the timing requirement after retiming. These values are then used to compute the delay weights of the nets in the ϵ network for retiming delay minimization. We note that the multilevel partitioning approach [25] is very effective in minimizing the weighted cutsize. Therefore, GEO performs multilevel partitioning to minimize the RTA-based weighted cutsize. However, RTA is done at the original netlist while a recursive multilevel approach performs partitioning on the subnetlist as well as its coarsened representations. Thus, it is crucial that we have an effective way to translate the RTA

```
DELAY-WEIGHT(NL')
set delay of edges in R (= retiming graph)
perform RTA (= retiming-aware timing analysis)
compute sequential slack for nodes in R
for each cluster C in NL'
    C(R) = all cells in R grouped into C
    slack(C) = min among cells in C(R)
X = top ε% clusters with small slack
for each net N in NL'
    if (all clusters in N are in X)
        compute delay weight of N
```

Fig. 5. Overview of the DELAY-WEIGHT algorithm that computes the weight of the nets in the ϵ network.

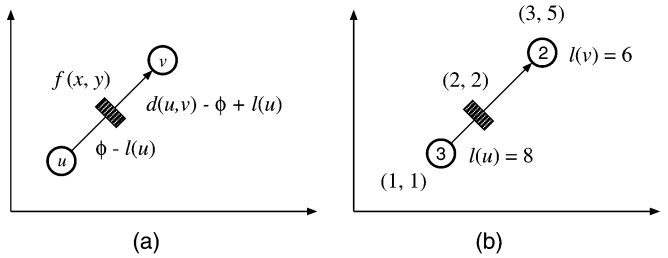

Fig. 6. Placement of $f \in$ FF along edge $e = (u, v)$. (a) f is placed at the position where the SAT equals to ϕ, (b) optimal location of f, where $\phi = 10$. f is placed at the point where the distance between u and f is $2 = \phi - l(u) = 10 - 8$.

results from the original netlist to the current coarsened subnetlist we are partitioning.

Fig. 5 shows DELAY-WEIGHT, our delay-weight calculator. Before we perform RTA, we initialize the edge delay in R (= retiming graph) based on the current global placement result—we set the delay of edges based on their Manhattan distance as discussed in Section II-A. Then, our Bellman–Ford variant RTA is performed from a given feasible delay to compute sequential slack. For each cluster C from the given coarsened subnetlist NL, we compute $C(R)$, the set of all the nodes in R that are grouped into C. We use the minimum slack among all cells in $C(R)$ as the slack for C. The reason we use the minimum slack value is because the critical path information is preserved regardless of multilevel clustering results (we have also performed experiments using average slack value instead of minimum. However, the minimum slack method generated better delay results). After the cluster slack computation is finished, we sort the clusters in a nondecreasing order of their slack values. We store the top ϵ% into a set X. For each net that contains only the clusters in X, we use the following equation to assign the *delay weight*:

$$\text{delay\_weight}(N) = \alpha \left(1 - \frac{\min\{\text{slack}(v) \mid v \in N\}}{\max\{\text{slack}(w) \mid w \in \text{NL}'\}}\right)^{\gamma}$$

where α determines the range of $\text{delay\_weight}(N)$, and γ helps to evenly distribute the weights between $[0, 1]$. This equation gives higher weights to the nets that contain smaller minimum cluster slack, thus giving higher priority to the nets containing more timing critical clusters. Instead of requiring all clusters in a net to be timing critical, we tried another scheme where we give delay weights to the nets with two or more timing critical clusters. Our related experiment indicates that this approach produced worse results. Our extensive experiments shown in Section V-B indicate that $\alpha = 10, \epsilon = 10$, and $\gamma = 3$ are an excellent empirical choice.

D. FF-Placement Phase

We note that retiming performs coarse-grained FF placement; it determines which edge gets which FF, but not the exact location on the edge. The optimal position of the ith FF computed by retiming on a path p is every point where the propagation delay equals to $i \cdot \phi$ for $1 \leq i \leq w^r(p)$. Thus, it is possible that the location is occupied by a cell. Under the nongeometric partitioning, we do not have a choice but to place f either at S_u or S_v for $f \in$ FF repositioned to $e(u, v)$ via retiming, where S_u denotes the slot u is assigned. However, finer-grained FF placement is possible under geometric embedding based partitioning since each cell is associated with certain geometric location. Note that the potential improvement can be as big as the delay of the global interconnect itself depending on the retiming result.

Our strategy to find out the exact location is as follows. We recompute the SAT for each vertex *after* retiming. Then, $l(v)$ for each vertex v represents the maximum delay from PIs to v. Thus, as depicted in Fig. 6, the optimal location of f can be determined based on $l(u)$ and the target delay ϕ: we move f from u toward v by a distance of $\phi - l(u)$. In other words, we find a position x where $l(x) = \phi$ on $e = (u, v)$. Assuming that x, u_x, and v_x, respectively, denote the x coordinate of f, u, and v (similar for y, u_y, and v_y), we obtain the following:

$$|x - u_x| : |v_x - x| = \phi - l(u) : d(e) - \phi + l(u)$$
$$|y - u_y| : |v_y - y| = \phi - l(u) : d(e) - \phi + l(u).$$

After rewriting these equations, we get

$$x = \frac{v_x \cdot [\phi - l(u)] + u_x \cdot [d(e) - \phi + l(u)]}{d(e)}$$
$$y = \frac{v_y \cdot [\phi - l(u)] + u_y \cdot [d(e) - \phi + l(u)]}{d(e)}.$$

Moreover, placement of FFs along the longest paths from PIs to POs will suffice in order to retime the circuit. We establish the following theorem as discussed in Section II-B.

Theorem 1: For a given geometric embedding-based partitioning solution P with a target delay ϕ, FF placement can retime P to a delay less than $\phi + D_g$ if the SATs for all POs less than or equal to ϕ.

Proof: The authors of [14] showed that P can be retimed to a delay less than $\phi + D$ if ϕ is feasible, where D denotes the maximum gate delay. The term D is from the fact that the optimal location may be occupied by a gate with the maximum delay. In this case, we place the FF right next to the gate, causing the delay to increase by D at the most. The same argument is applicable in PPR problem, since FF placement determines the optimal location of FFs along the interconnect. In case the optimal location is occupied by a gate with the maximum delay D_g, the final delay becomes $\phi + D_g$. If the FF placement is not used, the final delay is $\phi + D_e$, where D_e denote the maximum edge delay and $D_g \ll D_e$. □

Fig. 7 illustrates an impact of FF placement on the final delay result. $d(e), l(v)$, and $r(v)$, respectively, denote the edge delay, SAT, and retiming values. In Fig. 7(a), the FF is moved to a location closer to the source node of the global interconnect. The corresponding delay result is 11. This is what is typically done when performing retiming with no interconnect length information. In Fig. 7(b), our FF placement step finds a better FF location, where the FF is placed in the middle of the global interconnect. The corresponding delay result is 9.

We observe that the optimal positioning of FFs improve the final delay result obtained after retiming. Our related experiments shown in Table III from Section V-B indicate that the final retiming delay improves up to 15% for big sequential circuits. We emphasize that the finer-grained FF placement is possible only through the geometric embedding based partitioning. Finally, the complexity of GEO is that of construction phase and FF placement, which is $O(n \log n + k) = O(n \log n)$. Our experimental results in Section V-C confirm that the runtime of GEO indeed is comparable to that of other linear time algorithms such as hMetis [25].

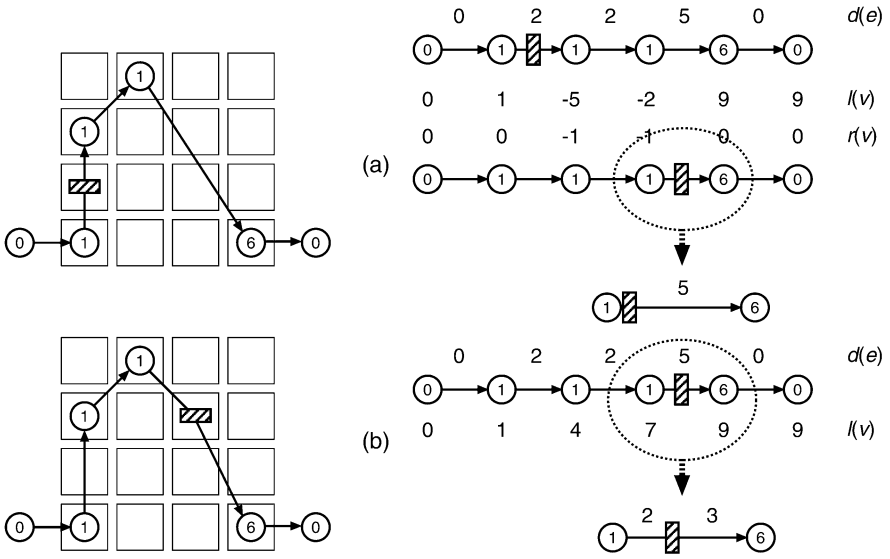

Fig. 7. Impact of FF placement on delay. $d(e), l(v)$, and $r(v)$ respectively denote the edge delay, SAT, and retiming values. (a) conventional FF placement with $\phi = 11$, (b) optimal FF placement with $\phi = 9$. $l(v)$ is recomputed after retiming.

V. Experimental Results

A. Experimental Setting

We implemented our algorithms in C++/STL, compiled with gcc v2.4, and tested on a SUN ULTRA SPARC60 at 360 Mhz. The benchmark set consists of six biggest sequential circuits (b14o–b22o) from ITC'99 [26], five from our industrial sponsor (ind1–ind5), and seven biggest from ISCAS'89 [27] (s13207–s9234). We chose ITC and ISCAS benchmark circuits since other benchmark circuits, such as ISPD, do not provide signal direction and gate delay information. ITC and ISCAS are the biggest circuits (in blif format) publically available at this moment. Detailed statistics of the circuits are shown in Table I.

We report delay, wirelength, and runtime based on 8×8 global placement, where each slot has a unit size of 1×1. Wirelength is the sum of half-perimeter of bounding box of nets. Runtime is in seconds. Edge delay is computed by the Manhattan distance between the two end points. For example, an edge connecting a gate at location (0, 0) and (7, 7) has a delay of 14. We assume that all gates have unit area and unit delay, while primary inputs and primary outputs have zero area and zero delay. FFs have unit area and zero delay. We obtained the latest binary executable of `hMetis` [25] (v1.5.3) and `HPM` [16] partitioners for comparison purpose. The bipartitioning area balance skew is set to (0.49, 0.51) for `hMetis` and `HPM` to enforce tight area balance among blocks.

Table I reveals the potential of retiming in further reducing the delay. Under the "STA" column, we perform STA while setting the delay of all edges to zero and report static delay. This serves as the lower bound on the static delay since global placement assigns nonzero delay to edges that connect gates in different location. Following the same reasoning, we report the lower bound of retiming delay under "RTA" column by performing our RTA while setting the delay of all edges to zero. The ratio between the lower bound of retiming and static delay $(=\text{retiming}/\text{static delay})$ is shown under the "R/S" column. We summarize our observation as follows:

- For most of the circuits, the delay improvement after retiming is significant—retiming enables up to 42% delay improvement for big sequential circuits such as "b21o" and "b22o." The average delay improvement is 23%.
- We note these lower bounds are based on gate delays only. Therefore, it is important to consider interconnects during global placement since the edge delay varies from 0 to 14 in case of 8×8 global placement. The conventional nongeometric algorithms that perform partitioning and retiming simultaneously [14]–[16] do not consider this wide range of edge delays and assume a single constant D for all interpartition edges.

TABLE I
Sequential Circuit Characteristics

sequential circuits			delay lower bound		
name	#GA	#FF	STA	RTA	R/S
b14o	5401	245	41	27	0.66
b15o	7092	449	45	38	0.84
b17o	22854	1414	44	38	0.86
b20o	11979	490	74	44	0.59
b21o	12156	490	74	43	0.58
b22o	17351	703	79	46	0.58
ind1	30273	617	286	286	1.00
ind2	28019	1759	39	27	0.69
ind3	52338	2008	468	462	0.99
ind4	102002	8335	76	72	0.95
ind5	31471	2397	271	165	0.61
s13207	8027	669	59	50	0.85
s15850	9786	597	82	62	0.76
s35932	16353	1728	29	27	0.93
s38417	22397	1636	47	32	0.68
s38584	19407	1452	56	47	0.84
s5378	2828	163	33	29	0.88
s9234	5597	211	58	38	0.66
AVE					0.77

B. STA Versus RTA

Table II shows the impact of different timing analysis engines on static and retiming delay results. Under the "STA" columns, we use STA to guide our global placement `GEO` and report both static and retiming delay results. Under the "RTA" columns, we use RTA to guide `GEO`. We report the ratio between RTA and STA-based results (=RTA/STA) under the "RTA/STA" columns. The last column "r/s" is the ratio between retiming delay of RTA and static delay of STA. Note that <STA+`GEO` targets static delay minimization while RTA+`GEO` performs retiming delay minimization. We summarize our observation as follows.

- STA+`GEO` achieves 11% an average improvement on static delay compared to RTA+`GEO`. This is expected since the focus of STA+`GEO` is on static delay.

TABLE II
COMPARISON BETWEEN STA-BASED AND RTA-BASED GLOBAL PLACEMENT

	STA+GEO		RTA+GEO		RTA/STA		
ckt	sta	ret	sta	ret	sta	ret	r/s
b14o	95	63	95	53	1.00	0.84	0.56
b15o	105	71	117	61	1.11	0.86	0.58
b17o	110	78	127	55	1.15	0.71	0.50
b20o	99	70	105	55	1.06	0.79	0.56
b21o	121	68	112	52	0.93	0.76	0.43
b22o	108	72	117	58	1.08	0.81	0.54
ind1	421	421	419	334	1.00	0.79	0.79
ind2	71	48	79	41	1.11	0.85	0.58
ind3	762	742	770	602	1.01	0.81	0.79
ind4	109	104	123	84	1.13	0.81	0.77
ind5	301	193	341	171	1.13	0.89	0.57
s13207	92	85	104	59	1.13	0.69	0.64
s15850	105	87	125	69	1.19	0.79	0.66
s35932	48	37	61	33	1.27	0.89	0.69
s38417	80	43	92	38	1.15	0.88	0.48
s38584	88	75	91	60	1.03	0.80	0.68
s5378	65	51	78	36	1.20	0.71	0.55
s9234	89	52	109	45	1.22	0.87	0.51
AVE	-		-		1.11	0.81	0.60
TIME	3142		12103		-		

TABLE III
IMPACT OF THE FF-PLACEMENT PHASE IN GEO ON DELAY IMPROVEMENT

	node type		FF placement			dly distr %	
ckt	bgn	end	bef	aft	a/b	gate	edge
b14o	FF	FF	60	53	0.88	26	74
b15o	FF	PO	63	61	0.97	23	77
b17o	PI	PO	55	55	1.00	15	85
b20o	PI	FF	58	55	0.95	22	78
b21o	FF	FF	61	52	0.85	32	68
b22o	PI	FF	67	58	0.87	34	66
ind1	FF	FF	354	334	0.94	15	85
ind2	FF	PO	41	41	1.00	24	76
ind3	PI	FF	608	602	0.99	18	82
ind4	FF	PO	90	84	0.93	29	71
ind5	FF	FF	190	171	0.90	31	69
s13207	FF	PO	64	59	0.92	17	83
s15850	PI	FF	70	69	0.99	24	76
s35932	PI	FF	35	33	0.94	13	87
s38417	FF	FF	42	38	0.90	17	83
s38584	PI	PO	60	60	1.00	18	82
s5378	FF	PO	37	36	0.97	41	59
s9234	FF	FF	52	45	0.87	38	62
AVE					0.94	24.3	75.7

- RTA+GEO achieves 19% an average improvement on retiming delay compared to STA+GEO. Therefore, STA+GEO produces better static delay results, while RTA+GEO produces better retiming delay results.
- The last column "r/s" reveals that the static delay STA+GEO produces is more than double the retiming delay RTA+GEO produces for most of the circuits. Thus, retiming remains very attractive in performance optimization, and our RTA based algorithm produces better retiming delay results compared to the conventional STA based algorithm.
- The runtime of RTA+GEO is four times higher than STA+GEO. This is due to the $O(n^2)$ worse case runtime complexity of our Bellman–Ford variant RTA algorithm. However, we observed that the sequential arrival and required times converge within a few iteration. Thus, the practical runtime is $O(k \times n)$, where k typically ranges from 5 to 50.

Table III shows the impact of FF placement phase in GEO on delay improvement. Under the "node type" columns, we report the type of the beginning and ending nodes in the critical path. Under the "FF placement" columns, we report the retiming delay before and after FF placement and their ratio (=after/before). Under the "dly distr %" columns, we report the percentage of the gate and edge delay on the overall critical path delay. We summarize our observation as follows.

- We observe from the "a/b" that FF placement can fine tune the location of FFs along the global edges and improve the retiming delay by 6% on the average. The delay improvement is as high as 15% in "b21o," one of the biggest circuits from ITC'99 [26] benchmark suite. The FF placement does not improve the delay of the critical paths that do not contain FFs.
- The "dly distr %" columns show that edge delay dominates the overall path delay. On the average, the edge delay is responsible for 75% of the overall path delay. This reveals the strong need for rigorous interconnect driven optimization.

TABLE IV
IMPACT OF THE PARAMETERS RELATED TO DELAY-WEIGHT COMPUTATION ON THE RETIMING DELAY

	alpha			epsilon		
ckt	5	10	20	5%	10%	20%
b14o	56	53	55	55	53	61
b15o	64	61	63	67	61	71
b17o	63	55	57	57	55	59
b20o	53	55	57	58	55	53
b21o	64	52	51	55	52	58
b22o	59	58	59	55	58	59
ind1	401	334	331	353	334	363
ind2	40	41	48	43	41	41
ind3	675	602	612	610	602	599
ind4	91	84	86	86	84	89
ind5	191	171	198	167	171	187
s13207	67	59	72	61	59	59
s15850	62	69	67	71	69	73
s35932	38	33	37	33	33	35
s38417	51	38	42	41	38	41
s38584	69	60	65	59	60	62
s5378	41	36	38	34	36	41
s9234	45	45	47	46	45	48
TOTAL	2130	1906	1985	1951	1906	1999

Table IV shows the impact of parameter tuning on the retiming delay. The "alpha" determines the range of integer-based delay weights, and "epsilon" determines the size of epsilon network as discussed in Section IV-C. If alpha is set to zero, GEO degenerates to cutize driven-mincut placement. From the results summarized in Table IV, we conclude that the value of 10 serves well for both alpha and epsilon. In order words, we pick the nets that connect the top 10% of cells with small slack and compute the delay weights. The range of these delay weights is (0, 10).

C. Comparison to the Conventional Approaches

We use hMetis and HPM partitioners to compare to GEO. hMetis and HPM performs partitioning only, so we need subsequent floorplanning to determine the location of these partitions. GEO is a min-cut-based global placer, so it does not need a separate floorplanning. We chose the polish expression-based slicing floorplanning [28] as opposed to the sequence pair-based nonslicing floorplanning [29], since $m \times n$ slot assignment is slicing-floorplanning.

Table V shows the comparison among: 1) hMetis [25] + SA (SA-based floorplanning[28]); 2) HPM [16] + SA; and 3) GEO. The first approach is a typical example of the conventional method, where partitioning is performed first to divide the circuit and the subsequent floorplanning determines the location of these partitions. The second approach combines partitioning and retiming and performs floorplanning separately. GEO combines all these steps for more effective delay improvement. For all three approaches, we report retiming delay, wirelength, and runtime based on 8×8 global placement results. We summarize our observation as follows.

TABLE V
COMPARISON AMONG 1) hMetis + SA-BASED FLOORPLANNING); 2) HPM + SA; AND 3) GEO

	hMetis+SA		HPM+SA		GEO	
ckt	dly	wire	dly	wire	dly	wire
b14o	61	2823	57	3671	53	3704
b15o	83	4823	71	5013	61	5118
b17o	50	8923	56	10032	55	9996
b20o	74	5132	68	6132	55	6069
b21o	58	5523	57	5721	52	5861
b22o	76	7012	63	7232	58	7313
ind1	423	4912	397	5324	334	5475
ind2	51	5932	40	6813	41	6756
ind3	787	10123	685	12323	602	13796
ind4	94	14235	89	16842	84	16524
ind5	179	3336	181	4012	171	3961
s13207	59	1323	63	1452	59	1583
s15850	76	1823	81	1723	69	1689
s35932	45	1602	35	1663	33	1666
s38417	43	2453	37	2453	38	2479
s38584	71	2745	63	2723	60	2556
s5378	33	1142	34	1384	36	1474
s9234	48	1435	45	1372	45	1380
TOTAL	2311	85297	2122	95885	1906	97400
RATIO	1.21	0.88	1.11	0.98	1.00	1.00
TIME	8742		11242		12103	

- After combining partitioning and retiming as in HPM + SA, we improve the retiming delay result of hMetis + SA by 10% on the average.
- We can further improve the HPM + SA result by 12% on the average with our GEO. This convincingly demonstrates the advantage of our geometric embedding-based unified approach over the conventional nongeometric embedding approaches for delay minimization.
- hMetis + SA obtains the best wirelength results, which are about 20% better than others. We note that the cutsize/wirelength advantage of hMetis results in positive impact on delay. However, more rigorous delay optimization through GEO improves the delay results further.
- We note that the runtime overhead of GEO is reasonable. Both hMetis + SA and HPM + SA need the subsequent SA-based floorplanning to find the location of the partitions, and SA-based approaches typically suffers from huge runtime and time-consuming parameter tuning.

VI. CONCLUSION AND ONGOING WORKS

In this paper, we proposed a unified approach to partitioning, floorplanning, and retiming for effective and efficient performance optimization. The integration enables partitioning to exploit the geometric delay model provided by the underlying floorplan. Simultaneous consideration of partitioning and retiming based on the geometric delay model enables us to hide global interconnect latency more effectively. We are currently trying to improve the wirelength results of GEO. In addition, instead of an expensive call to $O(k \times n)$ RTA engine for exact path analysis, we plan to perform incremental timing analysis for sequential circuits. Lastly, we plan to integrate buffer insertion into our RTA for delay and power minimization.

REFERENCES

[1] J. Cong, L. He, C. K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integration, VLSI J.*, pp. 1–94, 1996.

[2] C. J. Alpert, T. F. Chan, A. B. Kahng, I. L. Markov, and P. Mulet, "Faster minimization of linear wirelength for global placement," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 3–13, Jan. 1998.

[3] M. A. Breuer, "Class of min-cut placement algorithms," in *Proc. ACM Design Automation Conf.*, 1997, pp. 284–290.

[4] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 356–365, Mar. 1991.

[5] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. ACM Design Automation Conf.*, 1998, pp. 269–274.

[6] W. J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 349–359, Mar. 1995.

[7] S. Hur and J. Lills, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 165–170.

[8] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon 2000: Fast standard-cell placement for large circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 260–263.

[9] M. C. Yildiz and P. H. Madden, "Improved cut sequences for partitioning based placement," in *Proc. ACM Design Automation Conf.*, 2001, pp. 776–779.

[10] J. Cong, "An interconnect-centric design flow for nanometer technologies," in *Proc. Int. Symp. VLSI Technol., Syst., Applicat.*, 1999, pp. 54–57.

[11] E. L. Lawler, K. N. Levitt, and J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Trans. Computer-Aided Design*, vol. C-18, no. 1, pp. 47–57, 1966.

[12] R. Murgai, R. K. Brayton, and A. S. Vincentelli, "On clustering for minimum delay/area," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 6–9.

[13] R. Rajaraman and D. F. Wong, "Optimal clustering for delay minimization," in *Proc. ACM Design Automation Conf.*, 1993, pp. 309–314.

[14] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 489–498, June 1998.

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

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

[17] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, pp. 5–35, 1991.

[18] A. Ranjan, A. Srivastava, V. Karnam, and M. Sarrafzadeh, "Layout aware retiming," in *Proc. Great Lakes Symp. VLSI*, 2001, pp. 25–30.

[19] I. Neumann and W. Kunz, "Layout driven retiming using the coupled edge timing model," *IEEE Trans. Computer-Aided Design*, vol. 22, pp. 825–835, July 2003.

[20] J. Cong and D. Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc. Asia South Pacific Design Automation Conf.*, 1999, pp. 97–100.

[21] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 6, pp. 87–90, 1958.

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

[23] J. Cong and S. K. Lim, "Edge separability based circuit clustering with application to circuit partitioning," in *Proc. Asia South Pacific Design Automation Conf.*, 2000, pp. 429–434.

[24] ——, "Multiway partitioning with pairwise movement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1998, pp. 512–516.

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

[26] ITC 1999 Benchmark Suite [Online]. Available: http://www.cad.polito.it/tools/9.html

[27] ISCAS 1989 Benchmark Suite [Online]. Available: http://www.cbl.ncsu.edu

[28] D. F. Wong and C. L. Liu, "Floorplan design of VLSI circuits," *Algorithmica*, pp. 263–291, 1989.

[29] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle packing based module placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 472–479.