Physical Planning with Retiming*

Jason Cong and Sung Kyu Lim
UCLA Department of Computer Science, Los Angeles, CA 90095
{cong,limsk}@cs.ucla.edu

Abstract

In this paper, we propose a unified approach to partitioning, floorplanning, and retiming for effective and efficient performance optimization. The integration enables the partitioner to exploit more realistic *geometric delay model* provided by the underlying floorplan. Simultaneous consideration of partitioning and retiming under the geometric delay model enables us to hide global interconnect latency effectively by repositioning FF along long wires. Under the proposed *geometric embedding based performance driven partitioning problem*, our GEO algorithm performs multi-level top-down partitioning while determining the location of the partitions. We adopt the concept of *sequential arrival time* [14] and develop *sequential required time* in our retiming based timing analysis engine. GEO performs cluster-move based iterative improvement on top of multi-level cluster hierarchy [4], where the gain function obtained from the timing analysis is based on the minimization of cutsize, wirelength, and sequential slack. In our comparison to (i) state-of-the-art partitioner hMetis [9] followed by retiming [11] and simulated annealing based slicing floorplanning [15], and (ii) state-of-the-art simultaneous partitioning with retiming HPM [7] followed by floorplanning [15], GEO obtains 35% and 23% better delay results while maintaining comparable cutsize, wirelength, and runtime results.

1 Introduction

The conventional objective of partitioning is to minimize the number of connections among the subcircuits. Under the new interconnect-centric design paradigm, however, partitioning is seen as the crucial step that defines the local and global interconnects [2]. To meet the performance requirement of today's complex design, partitioners must consider the amount of interconnect induced by partitioning as well as its impact on performance. 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 [10, 13, 16], and for sequential circuits with retiming [14, 3, 7]. However, all of these works use a very simple delay model – unique delay value for gates, but single constant value for inter-block connection. This limitation is simply due to the fact that the location and dimension of blocks are not available during the partitioning.

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 total layout area is minimized. Traditionally, partitioning is performed first to generate blocks, followed by the subsequent floorplanning to determine the location of blocks. However, the separation between partitioning and floorplanning poses two major shortcomings for performance optimization; (i) partitioning suffers from non-realistic delay estimation as the location and dimension of blocks are not available, and (ii) the subsequent floorplanning may be constrained by the predefined local and global interconnects from the prior partitioning. Retiming [11] 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 but suffers from the lack of routing delay estimation. Most of existing algorithms on performance driven partitioning [10, 13, 12, 16, 5] consider only combinational circuits and thus ignore retiming. Recent advance in combining retiming with partitioning [14, 3, 7] enables the partitioner to explore wider solution space that considers equivalent FF movement. However, these algorithms impose a restriction on retiming of global interconnects – we cannot place FFs along wires but only at the beginning of the wires.

In this paper, we propose a unified approach to partitioning, floorplanning, and retiming for effective and efficient performance optimization. The integration enables the partitioner to exploit more realistic *geometric delay model* provided by the underlying floorplan, which in turn promotes more effective performance driven refinement. Simultaneous consideration of partitioning and retiming under the geometric delay model enables us to hide global interconnect latency more effectively – finer-grained FF placement is possible since each cell is associated with certain geometric location. Our GEO algorithm is based on multi-level framework, where the given solution is iteratively improved by cluster moves at each level of hierarchy [4]. We adopt the concept of *sequential arrival time* (SAT) [14] and develop *sequential required time* (SRT) in our retiming based timing analysis engine. The maximum SAT (MSAT) for a given partitioning solution translates into the delay result obtained after retiming, and our objective is to minimize both MSAT and cutsize. MSAT is used to derive SRT and its corresponding timing slack for each cell, and we use the slack information to guide cell move for effective delay and cutsize optimization. We demonstrate that our performance optimization becomes more effective based on the availability of geometry information from the underlying floorplan. In our comparison to (i) state-of-the-art partitioner hMetis [9] followed by retiming [11] and simulated annealing based slicing floorplanning [15], and (ii) state-of-the-art simultaneous partitioning with retiming algorithm HPM [7] followed by floorplanning [15], GEO obtains 35% and 23% better delay results while maintaining comparable cutsize, wirelength, and runtime results.

*This research was partially supported by the MARCO/DARPA Gigascale Silicon Research Center (GSRC) and a grant from Intel Corporation. The authors would like to thank Dr. Chang Wu and Dr. Peichen Pan at Aplus Design Technologies, Inc. for their helpful comments for this manuscript.

The remainder of the paper is organized as follows. Section 2 provides the problem formulation. Section 3 discusses our retiming based timing analysis engine. Section 4 presents our GEO algorithm. Section 5 provides experimental results. Section 6 concludes the paper with our ongoing research.

2 Problem Formulation

Given a sequential gate-level netlist $NL(C, N)$, let C denote cells consisting of gates and flip-flops, and N denote nets that connect cells. The purpose of Geometric Embedding based Performance Driven K-way Partitioning (GEPDP) is to assign cells in NL to K blocks while determining the location of the blocks under the prescribed area constraints. Given a GEPDP solution B, our primary objective is to minimize *delay* $\phi(B)$ (to be defined below). As secondary objectives, we minimize *cutsize* $\theta(B)$ and *wirelength* $l(B)$ induced by B. The formal definition of the problem is as follows.

Definition 1 – The GEPDP Problem
The geometric embedding based performance driven K-way partitioning problem under the given area constraints $A = (L_i, U_i)$ *for* $1 \leq i \leq K$ *has a solution* B, *where* $B = \{B_1(x_1, y_1), B_2(x_2, y_2), \cdots, B_K(x_K, y_K)\}$ *denotes the set of blocks and their locations.* B *is optimal if it satisfies the following conditions; (1)* $B_i \subset C$ *and* $L_i \leq |B_i| \leq U_i$ *for* $1 \leq i \leq K$, *(2)* $B_1 \cup B_2 \cup \cdots \cup B_K = C$, *(3)* $B_i \cap B_j = \emptyset$ *for all* $i \neq j$, *(4)* $\phi(B)$ *is minimized.*

2.1 Delay Objective

For delay calculation, we model NL with a directed, edge-weighted graph $G = (V, E)$, where the vertex set $V = \{v_1, v_2, \cdots, v_n\}$ represents cells, and the directed edge set $E = \{e_1, e_2, \cdots, e_m\}$ represents signal directions in NL. A directed edge $e(u, v)$ denotes the connection from vertex u to vertex v. In our *geometric delay model*, 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[1], i.e., $d(e) \propto |x_i - x_j| + |y_i - y_j|$, where $u \in B_i$ and $v \in B_j$. 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. The formal definition of $\phi(B)$ is as follows.

Definition 2 – Delay Without Retiming
The delay $\phi(B)$ *induced by a geometric embedding based partitioning solution* B *is the largest delay among the following 4 types of paths:* $\phi(B) = \max\{d(p(u, v)) | u \in PI \text{ or } FF \text{ and } v \in PO \text{ or } FF\}$.

We use *retiming graph* [11] for our retiming based timing analysis. A retiming graph $R = (V, E, W)$ consists of a vertex set $V = \{v_1, v_2, \cdots, v_n\}$ that represents gates, a directed edge set $E = \{e_1, e_2, \cdots, e_m\}$ that represents signal directions in NL, and edge weight set W that represents the the number of flip-flops (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 towards its inputs. A circuit is retimed to a delay ϕ by a retiming r if the following conditions are satisfied; (i) $w^r(e) \geq 0$, (ii) $w^r(p) \geq 1$ 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 result of partitioning.

For a given GEPDP solution B 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) of v, denoted $l(v)$, is the maximum path length from PIs to v. If the SAT for all POs are less than or equal to ϕ, the target delay ϕ is called feasible. Let $D_g = \max\{d(v) | v \in V\}$. The delay objective considering retiming is as follows.[2]

Definition 3 – Delay After Retiming
The delay $\phi(B)$ *induced by a geometric embedding based partitioning solution* B *after retiming for a given feasible target delay* ϕ *is defined to be the minimum* $\phi + D_g$.

2.2 Cutsize and Wirelength Objective

For the cutsize and wirelength calculation, we model NL with a hypergraph $H = (V, E_H)$, where the vertex set $V = \{v_1, v_2, \cdots, v_n\}$ represents cells, and the hyperedge set $E_H = \{h_1, h_2, \cdots, h_{m'}\}$ represents nets in NL. Each hyperedge h is a non-empty subset of V. The cutsize induced by B, denoted $\theta(B)$, is the number of hyperedges connecting vertices in different blocks. The x-span of a hyperedge h, denoted h_x, is defined as $h_x = \max_{c \in h}\{x_i | c \in B_i\} - \min_{c \in h}\{x_i | c \in B_i\}$. 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 B, denoted $l(B)$, is the sum of HPBB of all hyperedges.

3 Retiming based Timing Analysis

This section presents our retiming based timing analysis engine. We adopt the concept of *sequential arrival time* (SAT), which was first introduced in [14] and later on used in [3, 7] for partitioning with retiming. We extend SAT to introduce *sequential required time* and *sequential slack*. The slack information is used to derive ϵ-network that identifies timing critical cells in the circuit. In GEO, the derivation of ϵ-network plays a key role in determining the cell move gain.

3.1 Basic Concepts

The concepts of *arrival time*, *required time*, and *slack* are essential in timing analysis of circuits. The arrival time of a vertex $v \in V$ is defined to be the maximum path delay from PIs to v. For combinational circuits, we can compute arrival time of all vertices by visiting them in a topological order. We can assign the required time of a vertex $v \in V$ to specify timing constraint for v, i.e., the delay from PIs to v is "required" to be a certain value. Then, we can compute required time of all upstream vertices, i.e., all vertices reachable from PIs before arriving at v, by examining fan-out vertices. For combinational circuits, we can compute required time of all vertices by visiting them in a reverse

[1] The linear model is based on the assumption that various interconnect optimization is to be applied as a post-process. It is shown in [8] that this model is more realistic than the quadratic model.

[2] For the conventional non-geometric embedding based partitioning with $d(e) = D$ for all inter-block edges, the authors of [14] showed that B 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 GEPDP problem is $\phi + D_e$, where $D_e = \max\{d(e) | v \in E\}$. However, our FF placement phase in Section 4.3 reduces this bound to $\phi + D_g$, where $D_g << D_e$ according to Table 1.

topological order. The slack of a vertex $v \in V$ is defined to be the difference between the required time and arrival time of v. The timing slack is used to determine the timing criticality of vertices in V – the smaller the slack is, the more timing critical v is, and vice versa. The ϵ-network is defined to be a subgraph of R consisting of vertices whose slack is smaller than or equal to ϵ. Thus, ϵ-network consists of timing critical vertices that deserve attention for delay optimization.

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 sequential arrival time (SAT) of $v \in V$ in terms of fan-in vertices as follows;

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

In a similar way, we define the sequential required time (SRT) of $v \in V$ in terms of fan-out vertices as follows;

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

The slack of v, denoted $s(v)$, is given by $q(v) - l(v)$. Our Bellman-Ford variant shortest path algorithm presented in the next section uses these recursive definitions to derive the ϵ-network for given sequential circuits.

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 $r(v) = \lceil l(v)/\phi \rceil - 1$.

3.2 Timing Analysis Algorithm

Our retiming based timing analysis algorithm `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. Since R can possibly contain loops with positive weights depending upon target delay ϕ, `RTA` essentially performs single source longest path algorithm as in Bellmen-Ford algorithm [1]. 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 in Section 3.1.

The complexity of 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 [6] 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 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* vertex. 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 edges incident to them are set to zero.

```
RTA(R, φ)
Input: R(V, E, W) and target delay φ
Output: l(v), q(v), r(v), and π(v) for all v ∈ V
1.  for each vertex v ∈ V
2.      l(v) = r(v) = −∞;
3.      q(v) = ∞;
4.      π(v) = NULL;
5.  l(v_s) = r(v_s) = 0;
6.  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;
```

Figure 1: Description of `RTA` algorithm that computes the sequential arrival time $l(v)$, 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.

Figure 1 shows the description of our `RTA` algorithm. The initialization for sequential arrival time $l(v)$, sequential required time $q(v)$, retiming $r(v)$, and predecessor $\pi(v)$ for each vertex is done (line 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 fan-in (line 10) and fan-out (line 11) vertices. We also remember the fan-in 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 sequential arrival time of any PO is larger than the target delay ϕ, we conclude that ϕ is not feasible (line 13-14). Otherwise, if newly computed $l(v)$ is larger than the current value, we update $l(v)$, $r(v)$, and $\pi(v)$ (line 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 $q(v)$ is smaller than the current value (line 20-22). If `RTA` does not converge after $O(V)$ iterations, we conclude ϕ is not feasible (line 25).

4 GEO Partitioning Algorithm

We present our algorithm `GEO` in this section. We provide an overview of `GEO` algorithm, followed by the discussion of two main phases of `GEO`, *construction* and *FF-placement*.

4.1 Overview of GEO Algorithm

`GEO` is a multi-level geometric embedding based partitioner, which essentially performs a multi-level block placement in

a top-down manner. GEO generates blocks and determines their locations at each level of cluster hierarchy, where the number of blocks increases we go down the cluster hierarchy. GEO consists of two phases, namely, construction and FF placement phase. During the construction phase, multi-level partitioning with floorplanning is performed. First, a multi-level cluster hierarchy is generated, and the partitioning of top-level clusters is obtained. We use cell move based iterative improvement partitioning for the refinement of this initial solution. The gain value is determined by our retiming based timing analysis engine. The top-level partitioning information is then projected onto the next cluster hierarchy, and cell move is performed on the next level for further refinement. This top-down partitioning based on the multi-level cluster hierarchy continues until we obtain partitioning solution of the bottom level (= original circuit). 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 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.

4.2 Construction Phase

During the construction phase of GEO, multi-level cluster hierarchy is generated and the corresponding geometric embedding based partitioning is computed in a top-down manner. At each level of the multi-level cluster hierarchy, cell-move based iterative improvement partitioning is performed to improve the current partitioning solution. Figure 2 shows the description of the CONSTRUCT algorithm. The input to CONSTRUCT is a sequential netlist NL, the number of blocks desired K, and area constraint A. Then, CONSTRUCT divides cells in NL into K blocks, determines the location of blocks, and computes the corresponding minimum target delay. The height of cluster hierarchy is set to $\log_2 K$ (line 1) so that we perform 2-way partitioning on the top level, and 4-way partitioning on the next level, etc, until we perform K-way partitioning on the bottom level. In this way, we need to perform multi-level clustering only once for single K-way partitioning solution. We use ESC multi-level clustering algorithm [4] in GEO (line 2).

Let $B(i)$ denote $K/2^i$-way geometric embedding based partitioning of clusters on level i. Let R^i denote retiming graph of NL at level i, whereas R is the retiming graph of the original circuit. We generate a random initial 2-way partitioning and its corresponding minimum ϕ (line 3). We need to perform binary search to find the minimum ϕ, which requires $O(\log n)$ number of calls of RTA. This initial ϕ is used for all the subsequent call to RTA during CONSTRUCT. On each level i starting from the top ($= h-1$) to bottom level ($= 0$) (line 4), CONSTRUCT performs cell-move based iterative improvement partitioning. Depending on the current partitioning, each RTA call gives different ϵ-network. We perform RTA on the original circuit ($= R$) to obtain its ϵ-network R_ϵ (line 7). Then, we derive R^i_ϵ, the ϵ-network at the current level from R^i (line 8). We increase the weight of edges in R^i_ϵ (line 9), and the cell move gain is determined in such a way to minimize the *total weighted edge lengths* (line 10). By representing the cell move gain this way, we can use bucket

```
CONSTRUCT(NL, K, A)
Input: netlist NL, # of blocks K, and area bound A
Output: blocks B, location S, and min delay φ
1.  h = log2 K;
2.  ESC(h);
3.  obtain B(h − 1) and compute φ(B(h − 1));
4.  for (i = h − 1 downto 0)
5.      derive R^i;
6.      while (gain)
7.          R_ε = RTA(R, φ);
8.          derive ε-network R^i_ε;
9.          increase weights of edges in R^i_ε;
10.         move clusters on level i to improve B(i);
11.     if (i > 0)
12.         obtain B(i − 1) from B(i);
13. compute φ(B(0));
14. return B(0) and φ(B(0));
```

Figure 2: Description of CONSTRUCT algorithm that performs multi-level geometric embedding based top-down partitioning. $B(i)$ denotes $K/2^i$-way geometric embedding based partitioning of clusters at level i, and R^i denotes retiming graph of NL on level i.

structure to maintain linear time complexity during each pass. Since full timing analysis on R is costly ($= O(n)$), GEO performs RTA once per each pass of cell move. We obtain the next level partitioning from the current level (line 12). We randomly divide cells in each block into two and let CONSTRUCT improve this solution during the cell move of the clusters on the next level. At the bottom level, CONSTRUCT performs binary search once again to obtain the minimum feasible delay for the partitioning solution of the bottom level using RTA (line 13). Note that CONSTRUCT can generate multiple solutions and select the one that corresponds to the minimum ϕ to feed to the subsequent FF placement phase. Due to $O(\log n)$ number of calls to $O(n)$ time RTA, the complexity of CONSTRUCT is $O(n \log n)$.

4.3 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 i-th FF computed by retiming on a path p is every point where the propagation delay equals to $i \cdot \phi$ for $1 \le i \le w^r(p)$. Thus, it is possible that the location is occupied by a cell. Under the non-geometric partitioning, we do not have a choice but to place f either at B_u or B_v for $f \in FF$ repositioned to $e(u,v)$ via retiming. 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 sequential arrival time for each vertex *after* retiming. Then, $l(v)$ for each vertex v represents the maximum delay from PIs to v. Thus, as depicted in Figure 3, the optimal location of f can be determined based on $l(u)$ and the target delay ϕ: we move f from u towards 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 x-coordinate of f, u, and v (similar for y, u_y, and v_y), we obtain the following equations;

$$|x - u_x| : |v_x - x| \quad = \quad \phi - l(u) : d(e) - \phi + l(u)$$

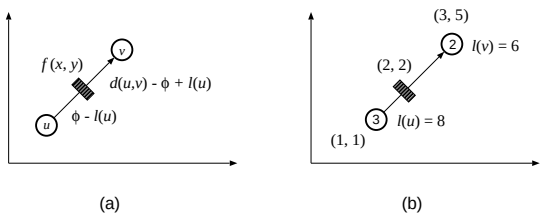

Figure 3: Placement of $f \in FF$ along edge $e = (u, v)$. (a) f is placed at the position where the sequential arrival time equals to ϕ, (b) optimal location of f, where $\phi = 10$. f is placed at the point where the distance between u and f is $2 = \phi - l(u) = 10 - 8$.

$$|y - u_y| : |v_y - y| \quad = \quad \phi - l(u) : d(e) - \phi + l(u)$$

After rewriting these equations, we get;

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

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

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 2.1.

Theorem 1 *For a given geometric embedding based partitioning solution B with a target delay ϕ, FF placement can retime B to a delay less than $\phi + D_g$ if the sequential arrival time for all POs are less than or equal to ϕ.*

Figure 4 illustrate impact of FF placement on delay. We observe that the optimal positioning of FFs improve the final delay result obtained after retiming. 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 `CONSTRUCT` and FF placement, which is $O(n \log n + k) = O(n \log n)$. Our experimental results in Section 5 confirm that the runtime of `GEO` indeed is comparable to that of other linear time algorithms such as `hMetis` [9].

5 Experimental Results

We implemented our algorithms in C++/STL, compiled with gcc v2.4, and tested on SUN ULTRA SPARC60 at 360Mhz. We obtained the latest binary executable of `hMetis` [9] (v1.5.3) for the evaluation. The benchmark set consists of 7 ISCAS circuits and 4 large scale industrial designs provided by our industrial sponsor. Detailed statistics of the circuits are shown in Table 1. We report delay, cutsize, wirelength, and runtime from 64-way partitioning embedded onto 8×8 grid. The grid length is computed by $\alpha \cdot \sqrt{|V|/64}$, where $\alpha = 1/14$. We obtain the constant α based on the assumption that the delay ratio among the gate, the shortest, and the longest edge is 1:3:40 in our biggest benchmark circuit *ind4* as shown in Table 1. The underlying chip dimension of $1cm \times 1cm$ for the current $.18\mu m$ technology is used for *ind4* [8]. Cutsize is based on cost-1 metric, and wirelength is the sum of half-perimeter of bounding box of nets. Runtime is in seconds and collected from ULTRA60 at 360MHz. The bipartitioning area balance skew is set to [.49, .51] for `hMetis` to enforce tight area balance among

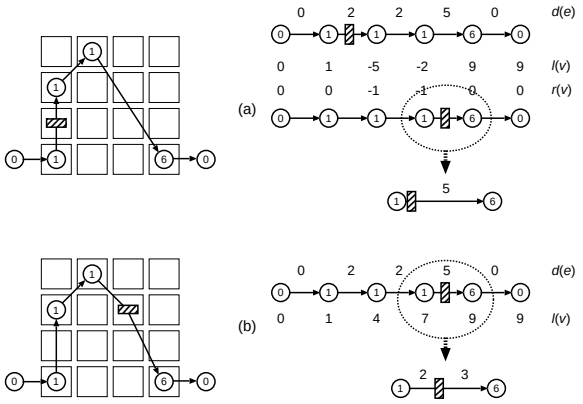

Figure 4: Impact of FF placement on delay. (a) conventional FF placement, where $f \in FF$ on $e = (u, v)$ is placed at B_u to give $\phi(B) = 11$, (b) optimal FF placement, where f is placed at the optimal location to give $\phi(B) = 9$. $l(v)$ is recomputed after retiming.

Sequential Benchmark Circuits					$d(e)$	
name	#GA	#PI	#PO	#FF	$\min_d$	max_d
s9234	1290	28	39	135	0.3	4.6
s5378	1443	35	49	163	0.3	4.8
s13207	3146	59	152	486	0.5	7.1
s15850	3784	76	150	515	0.6	7.8
bigkey	8599	228	197	224	0.8	11.8
s38584	13209	38	304	1423	1.0	14.4
clma	30552	61	82	33	1.6	22.0
ind1	29780	2630	3242	603	1.6	22.7
ind2	26060	2772	6242	1755	1.6	21.9
ind3	52197	2801	3070	2001	2.1	29.3
ind4	101531	4155	4547	8333	2.9	40.7

Table 1: Benchmark circuit characteristics. First 5 columns respectively denote the name, total number of gates, primary inputs, primary outputs, and flip-flops in each circuit, and next 2 columns respectively denote the shortest and longest edge delay $d(e)$, assuming gate delay is 1.

blocks. We assume that all gates have unit area and unit delay, while primary inputs, primary outputs and flip-flops have no area and no delay.

Table 2 shows the comparison among (i) `hMetis` [9] + retiming + floorplanning [15], (ii) `HPM` [7] + floorplanning [15], and (iii) `GEO`. The first approach is a typical example of the conventional method, where partitioning, retiming, and floorplanning are performed in a separate step in this sequence. The second approach combines partitioning and retiming and performs floorplanning separately. `GEO` combines all these steps for more effective delay improvement. We double the weight of edges in the ϵ-network for cell move gain computation (interested readers are referred to [6] for more detailed experimental results). After combining partitioning and retiming as in `HPM` + floorplanning, we improve the delay result of `hMetis` + retiming + floorplanning by 12%. However, we can further improve this result by 23% with our `GEO`. This convincingly demonstrates the advantage of our geometric embedding based unified approach over the conventional non-geometric embedding approaches for delay minimization. `hMetis` [9] + retiming + floorplanning obtains the best cutsize and wirelength results – about 20% better than others. The runtime overhead in `GEO` is reason-

	hMetis + Ret + Slicing				HPM + Slicing				GEO			
name	dly	cut	wire	time	dly	cut	wire	time	dly	cut	wire	time
s9234	27	268	4.2e3	587	24	512	6.2e3	612	22	642	6.2e3	24
s5378	26	337	5.4e3	512	24	632	8.4e3	475	18	710	8.2e3	23
s13207	48	353	8.2e3	528	47	496	1.2e3	634	40	451	1.1e4	43
s15850	57	465	1.1e4	660	53	694	1.4e4	712	48	671	1.5e4	71
bigkey	19	203	4.7e3	472	14	273	7.7e3	505	9	305	6.7e3	314
s38584	49	764	2.6e4	1364	46	824	3.2e4	1401	40	821	3.3e4	323
clma	46	820	5.2e4	1501	45	920	6.2e4	1677	31	941	6.2e4	1432
ind1	522	1657	1.1e5	2764	471	1734	1.6e5	3123	401	1869	1.5e5	3023
ind2	71	1249	1.1e5	1655	65	1954	1.5e5	2523	45	1520	1.5e5	812
ind3	1054	3311	3.3e5	3418	951	4023	3.9e5	4410	768	3951	3.8e5	4342
ind4	185	4031	4.4e5	5638	170	4932	6.2e5	6031	134	5014	6.2e5	6952
TOTAL	2104	13458	1.1e6	19099	1910	16994	1.4e6	22103	1556	16895	1.4e6	17359
RATIO	1.35	0.79	0.79	1.10	1.23	1.01	1.00	1.27	1.00	1.00	1.00	1.00

Table 2: Comparison among (i) `hMetis` [9] + retiming + floorplanning [15], (ii) `HPM` [7] + floorplanning [15], and (iii) `GEO`. 64 blocks are generated and embedded onto 8×8 grid. Delay is based on edge length $d(e)$ shown in Table 1. Cutsize is based on cost-1 metric, and wirelength is the sum of half-perimeter of bounding box of nets. Runtime is in seconds.

able – fastest among the algorithms used in this experiment.

6 Conclusions 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 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 cutsize and wirelength results of `GEO`. In addition, instead of expensive call to $O(n)$ retiming based timing analysis engine for exact path analysis, we plan to perform incremental timing analysis for sequential circuits.

References

[1] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, pages 87–90, 1958.

[2] J. Cong. An interconnect-centric design flow for nanometer technologies. In *Proc. of Int'l Symp. on VLSI Technology, Systems, and Applications*, pages 54–57, 1999.

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

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

[5] J. Cong and S. K. Lim. Performance driven multiway partitioning. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 441–446, 2000.

[6] J. Cong and S. K. Lim. Physical planning with retiming. Technical Report 200019, UCLA Computer Science Dept., 2000.

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

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

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

[10] E. L. Lawler, K. N. Levitt, and J. Turner. Module clustering to minimize delay in digital networks. *IEEE Trans. on Computer-Aided Design*, pages 47–57, 1969.

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

[12] L. Liu, M. Kuo, C. K. Cheng, and T. C. Hu. Performance driven partitioning using a replication graph approach. In *Proc. ACM Design Automation Conf.*, pages 206–210, 1995.

[13] R. Murgai, R. K. Brayton, and A. Sangiovanni Vincentelli. On clustering for minimum delay/area. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 6–9, 1991.

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

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

[16] H. Yang and D. F. Wong. Circuit clustering for delay minimization under area and pin constraints. *IEEE Trans. on Computer-Aided Design*, pages 976–986, 1997.