# Pyramids: An Efficient Computational Geometry-based Approach for Timing-Driven Placement *

Tao Luo[§†], David A. Papa[‡♯], Zhuo Li[♯],
C. N. Sze[♯], Charles J. Alpert[♯] and David Z. Pan[†]
§ Magma Design Automation / Austin, TX 78759
† University of Texas at Austin / Department of ECE / Austin, TX 78712
‡ University of Michigan / EECS Department / Ann Arbor, MI 48109
♯ IBM Austin Research Lab / 11501 Burnet Rd. / Austin, TX 78758
tluo@magma-da.com, iamyou@umich.edu, {lizhuo, csze, alpert}@us.ibm.com, dpan@ece.utexas.edu

## ABSTRACT

The purpose of global placement is to find non-overlapping locations for cells, typically while minimizing a wirelength objective. Because of this objective, however, when more timing information about the design is known, some cells will inevitably be sub-optimally placed from a timing perspective. In this paper, we present two new techniques to incrementally improve placements by moving cells to their optimal timing locations. We call our approach Pyramids, since it uses pyramid-shaped delay surfaces to solve for the optimal location, rather than running a more expensive linear programming solver. We show how to apply these techniques to timing-driven detailed placement and also for more accurate late-stage incremental timing correction. Experimental results validate the effectiveness of Pyramids by showing significantly improved timing after an industrial placement algorithm. Furthermore, compared to the linear programming solvers, the speedup of Pyramids solver is 373x vs. *CLP* and 448x vs. *GLPK*.

## 1. INTRODUCTION

Global placement is a well-studied optimization that seeks to find non-overlapping locations for every cell in the design, typically optimizing a wire length objective. As part of a physical synthesis flow, one also needs to satisfy timing constraints. Despite the best efforts, timing-driven global placement via net weights will never be a complete solution, since one cannot glean an accurate picture of the current timing until the placement has stabilized. Thus, there is a need for tools that can take an existing placement and incrementally modify it to improve its timing characteristics.

This problem has been exacerbated by technology scaling, since the increase in interconnect delay relative to gate delay means that cells placed far from their ideal timing locations suffer a greater timing penalty than in previous technology generations. Worst still, the emergence of multi-cycle paths requires careful latch/flip-flop placement to ensure balance of slacks for paths on both sides of the latch. Timing-driven global placement commonly uses net-weighting and net-constraints based methods to address timing [1, 2, 3], but they are again inadequate to solve the problem completely. Thus there is a body of work mostly using mathematical programming to incrementally improve circuit timing [4, 5, 6, 7, 8]. Mathematical programming based approaches can be expensive to solve. In response, previous applications focus on incremental placement of a limited number of objects to limit runtime cost.

This work presents a new physical optimization technique called Pyramids, designed to incrementally improve the timing characteristics of a layout via cell movement. Rather than using a generalized mathematical programming based framework [8], we develop special techniques to find the optimal solutions in timing-driven placement. The name Pyramids comes the fact that one can solve for the optimal set of locations by finding the intersection of a set of pyramid shaped delay surfaces emanating from the cells incident to the cells of interest.

By exposing and exploring the structure of the problem, Pyramids solves our linear programs with much lower computational complexity than that of a mathematical programming solver, which is designed to solve the generalized linear programs. In experiments, the LP solvers run many iterations for an optimal solution, while Pyramids computes the closed-form equations. In other words, the Pyramids is a specialized solver and is simpler and much faster than the LP solvers. Meanwhile, Pyramids finds the entire optimal region while existing linear programming solvers generate a single optimal solution [8]. Therefore, Pyramids generates a wider optimal region allowing more flexibility to avoid congestion in incremental placement. In brief, we make the following contributions.

- Pyramids can be used in several ways, two of which we describe in this paper. First, we show how the Pyramids solver can be used to perform timing-driven detailed placement via a bounding box capacitance model. One can iterate over critical cells in the design and move them to better locations very efficiently. This is effective for early timing-improvement of a global placer and could be embedded within a timing-driven placement algorithm.
- Later in a physical synthesis flow, one may require more accuracy and care to improve path delay. In this context we show how Pyramids can be used for critical path optimization under a linear delay model.
- Experiments show how effective these techniques are. The Pyramids based timing-driven detailed placer improves slack on placements optimized by the timing-driven placer in Cadence SOC encounter. For a large commercial ASIC, our second critical path optimization technique improves slack by more than 40% of the cycle time.
- Pyramids is an efficient solver for linear-programming-based formulations of timing-driven placement. In our experiments, comparing Pyramids to the open-source COIN-OR Linear Programming (CLP) solver [9], and the GNU Linear Programming Kit (GLPK) LP solver [10], Pyramids achieves a speedup of 373 and 448 times respectively.

*This work is supported in part by NSF, SRC, and IBM Faculty Award.

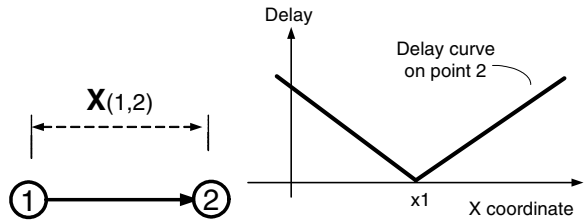
This paper presents two applications of the Pyramids solving technique, though Pyramids has other potential applications for modeling incremental timing-optimization techniques.

## 2. PRELIMINARIES AND FORMULATIONS

Most of signal nets are relatively short, local nets. For those short nets, gate delay can be modeled as a linear function of the capacitive load and input slew. For global and long nets, buffers have to be inserted to linearize the delay. The delay on an optimally buffered long net is approximately linearly proportional to its length [11, 12]. The traditional approach for timing optimization is to use a linear delay model for gates to calculate the circuit delay [4, 5, 6, 7, 8]. Static timing analysis is used to create linear programs that model the timing objective and a linear programming solver is used to generate an optimal location for each cell. However, LP based approaches are computationally expensive and not suitable for moving a large number of cells.

### 2.1 A Motivational Example

In the following, we show that the delay computations and the *max* and *min* operations in static timing analysis can be modeled as a geometry problem. It is possible to calculate the optimal locations of a cell by computing the intersections of curves and shapes, instead of using a generalized linear programming solver. Considering the following example. In Figure 1(a), point 1 is fixed and point 2 can move horizontally. Assuming the delay from point 1 to 2 is proportional to $x(1,2)$ – the distance between point 1 and 2. We plot the delay versus the location of point 2 in Figure 1(b), in which $x_1$ is the fixed $x$ coordinate of point 1. If we move point 2 to $x_1$, the delay to point 2 will be zero because the distance from point 1 is zero. If point 2 moves away from 1, the delay to point 2 grows linearly. The further we move point 2 away from 1, the larger the delay becomes – the result is a V-shaped curve, as shown in Figure 1(b).




(a) The delay propaga- tion between two points.

(b) The delay curve from point 1 to point 2.

**Figure 1: Plotting the delay vs. point position.**

Figure 2 shows the *max* operation between 2 V-shaped curves, D1 and D2. The larger of the curves between D1 and D2 is *max*(D1,D2). *Max* and *min* are basic operations in static timing analysis, and they are computed as the curve/surface intersections like the example shown in Figure 2. If a point moves horizontally and vertically simultaneously, the delay curves are no longer curves in a 2-D space, but surfaces in a 3-D space. The delay increments and *max*/*min* operations are computed by intersecting surfaces in a 3-D space. Therefore, it is possible to compute the optimal region/lines of a gate in timing driven placement through geometric computations.

By leveraging the above understanding of our problem, we can develop Pyramids solvers for different types of linear programs.

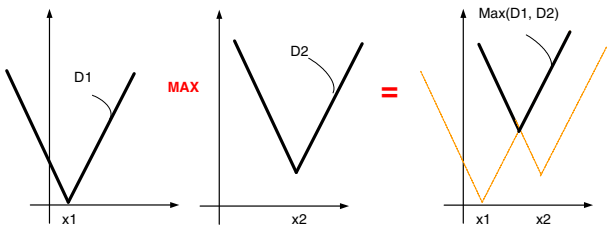



**Figure 2: The *max* operation of two curves.**

We show two Pyramids applications in different stages of the physical synthesis flow. In earlier stages of the flow, such as timing-driven global and detailed placement, where nets have not been optimally buffered, Pyramids uses the bounding box net model to estimate the capacitive load. In later stages of a physical synthesis flow, many buffers are added to decouple high fan-out nets. The Pyramids model decomposes the multi-terminal nets into 2-pin timing arcs. We formulate the bounding-box-model-based Pyramids timing-driven placement (Pyramids-DP) in Section 3, and timing-arc-model-based Pyramids timing refinement algorithm (Pyramids-CP) in Section 4. Currently, Pyramids is used to move a single gate. It is possible to extend Pyramids to move multiple gates.

### 2.2 Pyramids Formulations

The Pyramids algorithm identifies a movable gate $m$ and constructs a subcircuit. A subcircuit $G$ is a graph with the movable gate $m$ and its connected gates, as well as all signal nets connecting the gates. The problem is to maximize the minimum slack of the subcircuit by moving $m$ toward an optimal location. Static timing analysis is the basis of Pyramids timing optimization.

For a gate $g$ in subcircuit $G$, let $x_g$, $y_g$ denote the center coordinates of $g$. Let $L_n$ denote the Half Perimeter Wire Length (HPWL) of net $n$. Pin offsets are considered in implementation but not discussed in the rest of the paper for simplicity.

The unit capacitance constant is denoted by $c$, and the unit resistance constant is denoted by $r$. Let $Cap_n$ denote the total output capacitive load on net $n$. It is the sum of the wire capacitance of $n$ and the total pin capacitance driven by $n$, which is denoted by $Cpin_n$. Therefore, $Cap_n = cL_n + Cpin_n$.

The Required Arrival Time (RAT) on the inputs of a combinational gate $g$ is the minimum of the required arrival time propagated back from all gates driven by $g$.

$$RAT_g = \min_{0 \le i \le p} \{RAT_i - rc\mathrm{L} - D_g\} \quad (1)$$

The Actual Arrival Time (AAT) on the inputs of a combinational gate $g$ is equal to the maximum of the actual arrival time propagated from all drivers of $g$,

$$AAT_g = \max_{0 \le l \le k} \{AAT_l + rc\mathrm{L} + D_l\} \quad (2)$$

where $l$ is the driver of $g$, and L is the HPWL for the net driven by the driver gate.

The slack on the input of a gate $g$ is denoted by $S_g$, which is the difference between the required arrival time and actual arrival time

$$S_g = RAT_g - AAT_g \quad (3)$$

The AAT and RAT of the timing endpoints are generated by a static timer. Let $D_g$ denote the gate delay of $g$. The gate delay is a linear functions of the input slew, *Slew*, and total capacitive load, *Cap*. The coefficients are computed by fitting to the look-up-table-based standard-cell timing library. Here we show a simplified form of the delay equation that does not reflect the difference among pins. We

use different models for different pins in the implementation and the worst case model from the falling and rising transitions.

The gate delay $D_g$ is given by

$$D_g = d_I + a_g \cdot Slew_g + b_g \cdot Cap_g \quad (4)$$

where $a_g$ and $b_g$ are the fitting coefficients. $d_I$ denotes the intrinsic delay of the corresponding pin of the gate. We use a static input slew for delay computation, which is generated by a static timer.

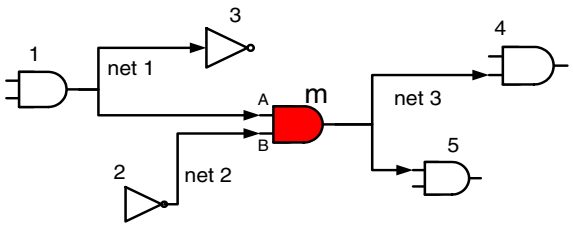



**Figure 3: A subcircuit with one movable gate.**

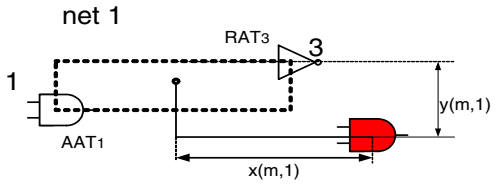



**Figure 4: The definition of the net length on net 1.**

# 3. PYRAMIDS ALGORITHM FOR TIMING-DRIVEN DETAILED PLACEMENT

The idea of the Pyramids algorithm is to transform a timing optimization problem into a geometric optimization problem. We refer the Pyramids algorithm for timing-driven detailed placement as Pyramids-DP in later sections.

Figure 3 is a simple subcircuit to illustrate the major steps in Pyramids-DP. In Figure 3, net 1 and net 3 are the multi-pin nets and net 2 is a two-pin net. Gate $m$ is selected as the movable gate, the movable gate could be a combinational or a sequential gate. All other gates connected with $m$ are considered fixed. For all gates in the subcircuit, the slew rate, required arrival time and actual arrival time have been computed by a timer. Assuming net $n$ is bounded by gate $p$ and gate $q$ in x dimension. Let $x_{(m,n)}$ denote the horizontal distance between $m$ and the geometric center of net $n$ with $m$ excluded. We have

$$x_{(m,n)} = |x_m - 0.5(x_p + x_q)| \quad (5)$$

Figure 4 shows the bounding box of the net 1 with $m$ excluded. For net 1 in Figure 4, $x_{(m,1)} = |x_m - 0.5(x_1 + x_3)|$ .

Let $y_{m,n}$ denote the vertical HPWL of net $n$. For net 1, $y_{(m,1)}$ is shown in Figure 4, and is constant in the formulation as we assume gate $m$ moves horizontally only.

## 3.1 Optimal Location Computation

Under a linear-delay model, the actual arrival time (AAT) on the input pin of a receiving gate $m$ is computed by adding the AAT on inputs of the driving gate $g$ to the delay due to capacitive load on the connection net $n$.

$$AAT_m = AAT_g + b_g cL_n + rcL_n \quad (6)$$

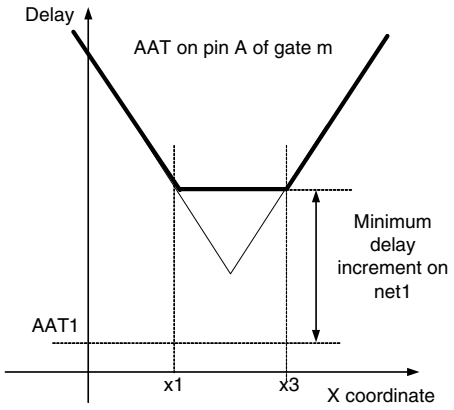



**Figure 5: The delay curve on the input of gate m.**

where $AAT_g$ is the actual arrival time on the input of gate $g$, and gate $g$ is the driver of net $n$. $b_g$ is the coefficient from Equation 4.

Let the vertical axis represent delay and the horizontal axis represent the $x$-coordinate. Figure 5 plots the actual arrival time on pin A of gate $m$, which is a function of the x coordinate of $m$, $x_m$. The HPWL of net 1 is $L_1 = L_1(y) + L_1(x)$. As gate $m$ moves horizontally, $L_1(y)$ is a constant number equal to $y_{(m,n)}$. $L_1(x)$ is the horizontal length of net $n$.

$$L_1(x) = \begin{cases} x_3 - x_1 & if x_1 \le x_m \le x_3; \\ x_m - x_3 + (x_3 - x_1) & \text{if } x_m > x_3 \\ x_1 - x_m + (x_3 - x_1) & \text{if } x_m < x_1. \end{cases} \quad (7)$$

where $x_1$ and $x_3$ are the $x$-coordinate of gate 1 and 3 in Figure 3. Therefore, the AAT curve on pin A of gate $m$ is determined by the geometric intersection of following set of linear equations.

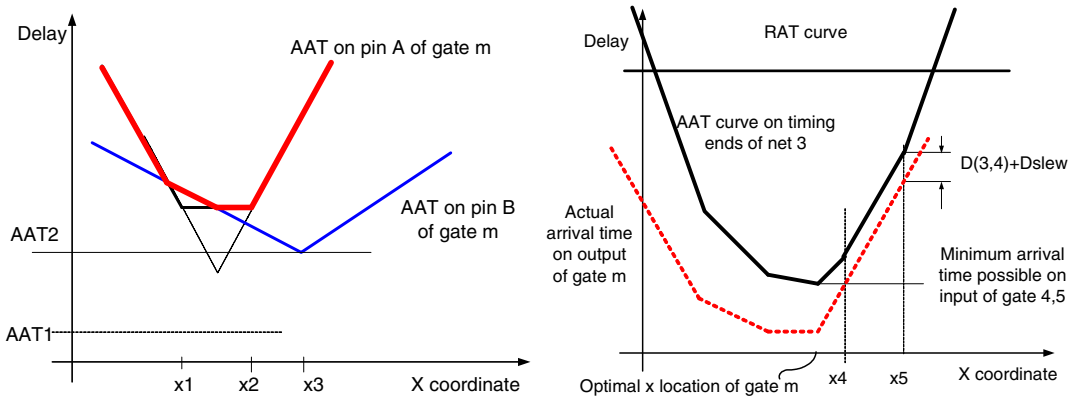
$$AAT = AAT_1 + c(b_1 + r)(y_{(m,1)} + x_3 - x_1) \quad (8)$$

$$AAT = AAT_1 + c(b_1 + r)(y_{(m,1)} + x_{(m,1)} + 0.5(x_3 - x_1)) \quad (9)$$

Note that Equation 8 is the horizontal line in Figure 5. As the input slew of gate 1 is assumed to be a constant value, in the above equations, the delay due to input slew is not shown for simplicity. In Figure 5. The slopes of the delay curve are $\pm c(b_1 + r)$, which capture the driving strength of gate 1.

The delay curve on pin B of gate $m$ is the V-shaped curve shown in Figure 6(a). The AAT on a combinational gate $m$ is computed by adding the greater of the AATs on input pins of $m$ with the gate delay on $m$. By intersecting the V-shaped curves with the delay curves on net 1 and taking the supremum, we get the solid delay curve segments in Figure 6(a), which represent the AAT on the inputs of $m$.

Figure 6(b) shows the AAT curve generation on the input of gate 4 and gate 5. $x_4$ and $x_5$ are the $x$-coordinates of gate 4 and 5 respectively. The AAT on timing end points of net 3 are computed by adding the worst AAT on the inputs of gate $m$, the delay on gate $m$, to the delay on net 3. Both the gate delay and net delay are linear functions of $x_m$. Therefore, we add up the delay curves on gate and net to the AAT curve segments of gate $m$ to generate the arrival time curve on net 3, shown as the solid curve segments in Figure 6. The horizontal line in Figure 6, represents the worst case RAT on the corresponding input of gate 3 and 4. The difference between the RAT and AAT curve is the slack curve, and the top of slack curve is the best slack achievable by moving gate $m$ horizontally. In this example, the optimal $x$ location of gate $m$ is shown in Figure 6, given a specified $y$ location. If $m$ is a sequential gate, the AATs




(a) The delay curve on on inputs of gate $m$.

(b) The delay curve on on net 3.

**Figure 6: The delay curves for subcircuit in Figure 3.**

on the input of gate 4 and gate 5 are constant and the computations are much easier.

## 3.2 Pyramids-DP Algorithm

In the following, we discuss the application of Pyramids algorithm in timing-driven detailed placement. The input of Pyramids is a legal placement. The placement is analyzed by a static timing analysis tool and the timing information is annotated in the placement database. The Pyramids-DP collects a few critical gates and sorts them by the worst slack on the pins. Pyramids starts with the worst gate and works on each gate in the queue order. Once a gate is moved the timing information is incrementally updated. The above process is repeated to a desired extent.

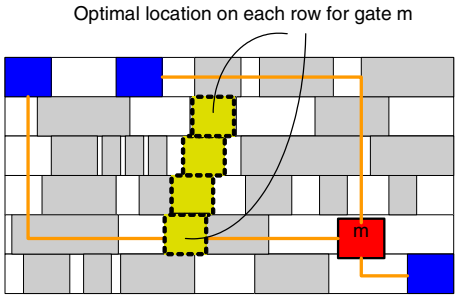



**Figure 7: The optimal $x$ location for the movable in each row in the bounding box are computed, and the optimal slack to assign the movable in each row is recorded.**

In each optimization iteration, once a movable gate is selected, Pyramids analyzes the subcircuit and rejects the gate if the bounding box of the the subcircuit is too small, which implies that it is unlikely to get any improvement. The bounding box of the candidate subcircuit covers a set of rows. Starting from the middle row of the subcircuit bounding box, the movable gate is assigned a row location, and Pyramids computes the optimal $x$ location, as shown in Figure 7. Then rows up and rows down are selected to compute the optimal location. All slack improvement scores are recorded. Pyramids-DP does not need to compute the score of all rows. During the experiments, the optimal $x$ location is the same for each row on most of cases.

Each row of the placement is divided into bins. The bin width is set to several times of the width of an average gate, and bin density is computed. The top row is selected and if there is enough space, the movable cell is placed. Otherwise, the overlapped gate is tentatively move to a neighboring bin with lowest density. If the worst slack of that gate becomes negative, such a swapping is rejected and next row candidate is considered. Otherwise the gate assignment is accepted. A simplified internal timer computes the slack changes during the swapping. The algorithm is shown in Algorithm 1. All gates are within rows when the timing optimization procedure terminates. There will be a small amount of residual overlaps in the $x$ dimension that will be removed by a legalizer.

**Algorithm 1** Pyramids-DP(Timing driven detailed placement flow)

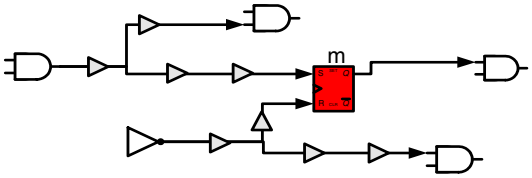
```
1:  PyramidsDP(critical gates Queue)
2:  foreach gate in Queue
3:      Build the subcircuit, check if the subcircuit can be improved
4:      foreach row in candidate rows
5:          slackScore = ComputeOptimalLocation(subcircuit, row)
6:      end foreach
7:      Sort rows on slack improvement score
8:      foreach row in the sorted rows
9:          Move the overlap gate to the low density neighbor bin
10:         if Slack is negative, accept change break
11:     end foreach
12: end foreach

13: ComputeOptimalLocation(subcircuit, row)
14:     Compute the delay lines on inputs of the gate
15:     if (movable is a latch/FF) then
16:         Compute the delay planes on all inputs
17:     else
18:         Intersect input delay lines to find the top line segments
19:         Grow output delay lines on top of input delay line segments
20:     end if
21:     Compute all slack lines based on the RAT
22:     Intersect all slack lines and find the bottom line segments
23:     Mapping the top point of the line segments to x-axis
24:     end if
25:     return optimal x and min-slack
```

# 4. PYRAMIDS ALGORITHM FOR CRITICAL PATH REFINEMENT

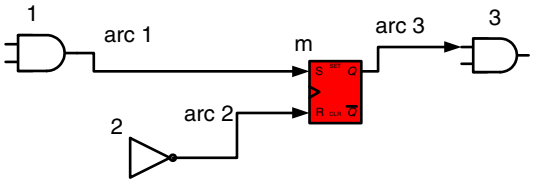
During later stages in a physical synthesis flow, Pyramids moves a few gates, especially imbalanced latches, to further improve timing. We refer the Pyramids algorithm for critical paths refinement

as Pyramids-CP in the following. Imbalanced latches are those with positive slack on one side and negative slack on the other side. Obviously, imbalanced latches present an opportunity to improve timing by moving the latch toward the negative side. However, in later physical synthesis stages, many buffers are added to decompose high fan-out nets and linearize the delay on long nets. Using a bounding box net model for the multi-pin nets is no longer suitable.

When a subcircuit contains buffered nets, the movement of gates is unnecessarily restricted. In addition, whenever a gate is moved the connected buffers have to be considered properly to avoid timing degradation. In the example shown in Figure 4, moving $m$ closer to the input drivers will reduce the negative slack. The ideal solution is to move the latch and buffer the nets simultaneously. However, multi-objective optimizations are often computationally prohibitive and non-trivial to realize. Pyramids-CP computes the optimal location of the movable gate based on timing estimation that considers buffers that will be inserted in the future. Once the gate is moved, all the nets in the subcircuit may be buffered again. This approach has been proved effective and efficient by experiments, and is easily integrated into a typical physical synthesis flow. To accommodate this approach, the following timing model is critical for Pyramids-based timing refinement.




(a) A subcircuit with buffers.




(b) The new timing model.

**Figure 8: The net model in Pyramids-CP.**

## 4.1 Linear Buffered-Path Delay Estimation

Buffering can not be ignored during the later physical synthesis stages for interconnect delay estimation [13, 14, 12]. Therefore, the interconnect delay model must be aware of the buffers that will be inserted in the future. We found that a linear delay model [11, 12, 8] is most appropriate. Therefore all multi-pin nets are represented by two-pin timing arcs that connect drivers to sinks. As shown in Figure 8, directly related timing arcs are included in the subcircuit. In this model, the delay along an optimally buffered interconnect is

$$d(L) = L(cR_b + rC_b + \sqrt{2R_bC_brc}) \tag{10}$$

where $L$ is the length of a two-pin buffered net, $R_b$ and $C_b$ is the intrinsic resistance and input capacitance of buffers and gates while $r$ and $c$ are unit wire resistance and capacitance respectively.

Empirical results in [12] indicate that Equation 10 is accurate up to 0.5% when at least one buffer is inserted along the net. Furthermore, our own empirical results suggest the model is reasonable enough to justify the latch location. The details are described in section 5.2.

## 4.2 Pyramids Refinement Algorithm Formulations

In this formulation, $\tau$ is a technology dependent parameter that is equal to the ratio of the delay of an optimally-buffered, arbitrarily-long wire segment to its length

$$\tau = delay(wire)/length(wire) \tag{11}$$

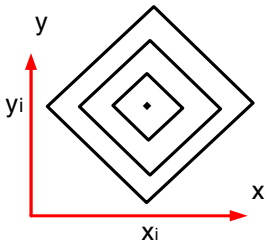
A timing arc is specified for a given net $n$ driven by gate $u$ and having sink $v$. We use the following definition of the required arrival time on the input of a combinational gate $g$, which has a similar form as equation 1.

$$RAT_g^{'} = \min_{0\le i\le p}\{RAT_i^{'} - \tau L - D_g^{'}\} \tag{12}$$

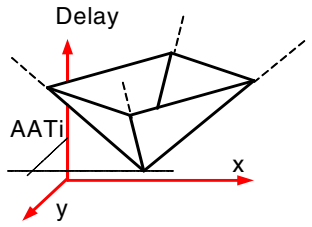
The actual arrival time on the output of a combinational gate $g$ is

$$AAT_g^{'} = \max_{0\le j\le k}\{AAT_j^{'} + \tau L + D_j^{'}\} \tag{13}$$

Different from Pyramids-DP, L is the HPWL for the timing arc between the driver and receiver, and the delay here is mainly modeled on the timing arcs. Furthermore, the gate delay $D^{'}$ is a constant value.




(a) The Manhattan contours.




(b) The delay surface: an upside down Pyramid.

**Figure 9: The shape of delay surface in 2d and 3d spaces.**

Pyramids-CP optimizes the following linear program, which is to optimize the worst slack in a subcircuit.

$$\begin{aligned} &OBJECTIVE: \quad Maximize \quad S \\ &st. \quad S \le RAT_v^{'} - AAT_u^{'} - \tau L_n \\ &\qquad \text{For every timing arc } n \text{ in the subcircuit.} \end{aligned} \tag{14}$$

The above linear program is essentially the linear program for single cell optimization in RUMBLE [8]. In the following, we show how to solve the above linear program using Pyramids.

## 4.3 Compute Optimal Location in Pyramids-CP

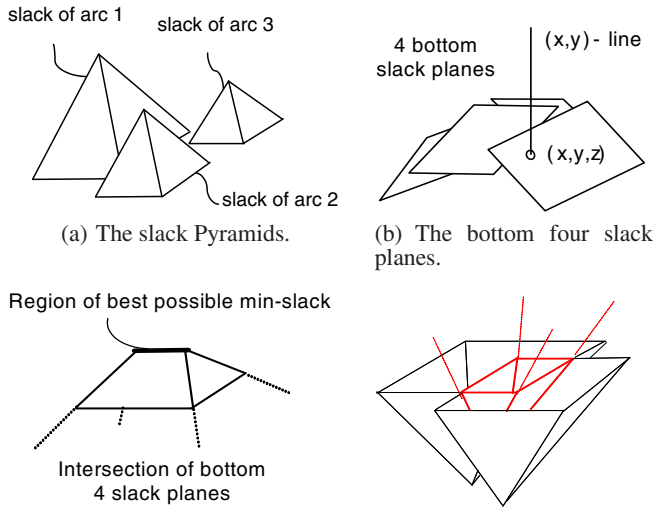
Similar to Pyramids-DP, the RATs, AATs and slacks are linear equations in terms of the cell coordinates that can be solved by a geometry-based approach. Here we assume a gate moves in $x$ and $y$ dimensions simultaneously, we need a 3-D space for geometric operations. The AAT on a timing arc is a linear equation of the Manhattan distance between driver gate and receiver gate of a timing arc. This value can be computed by intersecting delay planes in a 3-D space. Let $(x_g, y_g)$ denote the coordinate of gate $g$. For a timing arc driven by $g$ and with a sink $m$, the delay on the input of $m$ is determined by the following equations.

$$Delay = AAT_g + \tau(|x_m - x_g| + |y_m - y_g|) \tag{15}$$

The above equation determines four planes in 3-D space that intersect at a single point. The resulting shape is similar to an inverted pyramid in 3-D space. We named our algorithms Pyramids due to the shape of this delay surface. To illustrate the concept, the contours of the pyramid delay surface are shown in Figure 9(a), and a 3-D projection of the delay surfaces are shown in Figure 9(b).

### 4.3.1 Optimize the Sequential Gate

If the movable is a sequential gate, as in Figure 8, the RATs on timing arcs are constant. In other words, the RAT is a plane parallel to the *x-y* plane in 3-D space. The difference between the RAT and the AAT surface is the slack surface, which is pyramid-shaped. In the example in Figure 8, there will be twelve slack surfaces generated in total, for timing arcs 1, 2 and 3, as shown in Figure 10(a). There are four planes for each slack pyramid, and all slack surfaces can be categorized into four groups. The slack surfaces in the same group are parallel to each other.




(a) The slack Pyramids.

(b) The bottom four slack planes.

(c) The intersection of bottom slack planes.

(d) The delay surface on the output of the movable gate.

**Figure 10: Computation of the optimal region in Pyramids-CP.**

To find the best slack surface possible, two testing points, such as $(0,0)$ and $(0, chip-y-dimension)$ are sufficient to find the orderings of all slacks surfaces. The idea is illustrated in Figure 10(b), and there will be four least-slack planes. The intersection of four bottom planes form a "trough"-shaped polyhedron, which is the best possible worst-slack region of the subcircuit. Figure 10(c) illustrates trough-shaped slack surface for this example. Mapping the top line segment of the trough to the placement region is the slack-optimal region for the movable. Moving the latch to any point on the line segment achieves the best possible worst slack.

### 4.3.2 Optimizing Combinational Gates

If the movable is a combinational gate, the AAT on the output of the gate is the maximum of the AATs on the inputs of the movable adding the gate delay. The *max* operation of the delay surfaces can be computed in similar fashion to the method for computing the least-slack planes in section 4.3.1. Still, two testing points are sufficient to find the top four delay planes, as shown in Figure 10(d).

The delay pyramid of each output of the movable gate "grows" on top of the actual arrival time trough of the movable, which is a basin-type shape having nine delay planes with the bottom plane parallel to the *x-y* plane. As the RAT on each output timing arcs is a constant plane parallel to the *x-y* plane. The bottom slack planes form a reverse basin similar to that formed by the delay planes. Note only the 9 bottom planes are required for finding the entire optimal region. The implementation and computation is simplified if just one optimal point or a portion of the optimal region are needed, as the the optimal region intersects with bottom of the input delay trough.

### 4.3.3 Pyramids-CP Algorithm

In this section, we discuss Pyramids-CP for timing refinement, which mainly focuses on improving poorly placed latches, similar to [8]. Once an imbalanced latch is selected, Pyramids selects a subcircuit and a snapshot is cached which includes previous buffer locations, signal polarities, and the latch location. It then removes all buffers in the subcircuit. Pyramids-CP then improves the timing adhering to a "Do no harm philosophy". In the Pyramids framework, nets are re-buffered after the latches have been moved, and the actual impact of the latch movement is evaluated by an incremental timer. If the timing is not improved due to various reasons, such as blockages, poor buffering, etc., the changes are rejected and previous status of the subcircuit is restored.

The Pyramids-CP algorithm is shown in Algorithm 2.

**Algorithm 2** The Pyramids-CP (For critical paths refinement)

```
1:  PyramidsAlgorithmII(movable gate)
2:      Identify the subcircuit
3:      Rip-up buffers and save buffers in cache
4:      Compute the delay planes on inputs of the gate
5:      if (movable is a latch) then
6:          Compute the delay planes on all inputs
7:      else
8:          Find the top delay planes by ordering all input delay planes
9:          Grow output delay planes on top of input delay trough
10:     end if
11:     Compute all slack planes based on the RAT and delay planes
12:     Find the bottom slack planes by ordering all slack planes
13:     Find the optimal region by intersecting the bottom slack planes
14:     Move the gate a suitable optimal location
15:     Re-buffer the subcircuit
16:     Evaluate the subcircuit timing by incremental timing analysis
17:     if Timing is improved
18:         Keep the change, release the cache
19:     else
20:     Drop the change, restore the subcircuit from the cache
21:     end if
```

## 5. EXPERIMENTS

| *design* | *cells* | *Regs* | *Pins* | *nets* |
|---|---|---|---|---|
| s838_1 | 404 | 32 | 37 | 404 |
| s1196 | 513 | 18 | 30 | 500 |
| s1238 | 616 | 18 | 30 | 603 |
| s1423 | 631 | 74 | 24 | 627 |
| s1488 | 665 | 6 | 29 | 647 |
| s1494 | 672 | 6 | 29 | 654 |
| s15850 | 788 | 165 | 38 | 701 |
| s9234_1 | 1051 | 145 | 67 | 1006 |
| s13207 | 1373 | 333 | 76 | 1235 |
| s5378 | 1380 | 163 | 83 | 1332 |
| s38584 | 7016 | 1178 | 233 | 6741 |
| s35932 | 7630 | 1728 | 69 | 7311 |
| s38417 | 8414 | 1564 | 136 | 8309 |

**Table 1: The characteristics of the ISCAS benchmarks.**

We have implemented Pyramids-DP and Pyramids-CP in C++. Pyramids-DP is based on OpenAccess (OA) [15, 16], which is an open source database system for EDA applications. Pyramids-CP is implemented within an industrial physical synthesis flow. For Pyramids-DP, We have adopted the open source, incremental static timing engine in the OA Gear package [17], as the timing analysis tool. There is a hypothetical standard cell library provided in OAGear to resemble a typical 180nm process. We have linearly scaled the 180nm library down to 130nm, 90nm, 65nm, and 45nm. For each technology node, the library parameters are scaled by an 80% factor to its previous technology generation. We use the 45nm library as the timing library for all OA based experiments. For benchmarks, we select the largest circuits in the ISCAS89 sequential logic benchmarks, and convert the netlist into OA format. The characteristics of the benchmarks are shown in Table 1.

## 5.1 Pyramids-DP Experiments in OpenAccess Environment

We tested Pyramids-DP on all circuits in Table 1. We use Cadence 2005 SOC encounter timing driven placer to generate the initial placements. The OAGear Timer is used to measure the worst case delay of the placement generated by Cadence timing driven placer. For each circuit, the timing target is set to a value about 95% of initial worst case delay. Pyramids-DP reads the initial placement and improves the timing further. Table 2 reports the further improvement accomplished by Pyramids-DP. Column 8 shows the target timing for each circuit. Columns 2-4 report the worst negative slack (WNS), total negative slacks (TNS) and the wire length (WL) of the initial placements. Columns 5-6 report the same of the Pyramids-DP improved results. Columns 9-11 report the Pyramids-DP improvement by percentage. Compared with the initial placement optimized by Cadence timing driven placer. The WNS and TNS are improved by 30.4% and 46.7% on average on the pre-set timing target, with little length increase (WL).

In next set of experiments, we show why latch placement is important for timing. We fix the location of all sequential cells in the initial placements and run Pyramids-DP on combinational gate only. The results are report in Table 3. Therefore, Table 3 reports how much timing improvement the Pyramids-DP is able to achieve by moving combinational cells only. Clearly, without moving the latches, the amount of timing improvement achievable is much less.

| Results with Fixed Sequential Cells. | | | | |
|---|---|---|---|---|
| Circuits | Change (%) | | | CPU |
| | *WNS* | *TNS* | *WL* | (*s*) |
| s838_1 | 17.3 | 35.9 | -0.4 | 4.6 |
| s1196 | 10.5 | 17.6 | 1.0 | 1.7 |
| s1238 | 44.8 | 64.5 | 0.4 | 4.0 |
| s1423 | 32.1 | 46.5 | -0.0 | 10.7 |
| s1488 | 24.4 | 44.1 | 0.6 | 5.5 |
| s1494 | 28.7 | 56.1 | 0.5 | 3.3 |
| s15850 | 8.2 | 20.3 | -1.4 | 9.4 |
| s9234_1 | 9.6 | 29.8 | 4.1 | 17.4 |
| s13207 | 2.2 | 6.6 | -0.5 | 26.3 |
| s5378 | 11.1 | 20.4 | 1.2 | 11.5 |
| s38584 | 4.3 | 4.7 | -0.1 | 51.0 |
| s35932 | 7.4 | 21.2 | 0.1 | 90.7 |
| s38417 | 14.2 | 29.7 | 0.2 | 67.5 |
| avg | **16.5%** | **30.6%** | 0.4% | |

**Table 3: Pyramids-DP results with fixed sequential cells.**

## 5.2 Pyramids-CP Experiments in an Industrial Flow

In the following, we show the experimental results for Pyramids-CP within an industrial physical synthesis flow. First, we show how well our model resembles the realistic timing. Table 4 compares the model slack predicted to values measured by a commercial static timing analyzer. The latches are moved and nets are buffered before the measurement. Columns 2-4 report the initial, final, and improvement in worst slack, with the model presented in Section 4. The timing measured by STA engine is reported in Columns 5-7. The observation is that there is over 90% correlation between the model's predicted improvement and timing values measured with an industrial timing engine, which implies the model is accurate enough to guide optimization.

| Model timing vs. reference timing | | | | | | |
|---|---|---|---|---|---|---|
| | Model slack (ps) | | | Subcircuit slack (ps) | | |
| Subcircuit | orig | new | imprv. | orig | new | imprv. |
| latch A0 | -1171 | 201 | 1373 | -945 | 317 | 1263 |
| latch A1 | -1147 | -527 | 620 | -953 | -390 | 563 |
| latch A2 | -1081 | -271 | 810 | -962 | -227 | 735 |
| latch A3 | -958 | -36 | 923 | -706 | 123 | 828 |
| latch A4 | -920 | 320 | 1241 | -690 | 395 | 1085 |
| latch A5 | -964 | 296 | 1260 | -681 | 376 | 1058 |
| latch A6 | -924 | 405 | 1328 | -679 | 351 | 1030 |
| latch A7 | -913 | 213 | 1126 | -633 | 290 | 923 |
| latch A8 | -876 | 342 | 1218 | -614 | 440 | 1054 |
| latch A9 | -800 | 397 | 1198 | -610 | 262 | 872 |
| avg | -887 | 122 | 1009 | -679 | 176 | 856 |

**Table 4: The model used in Pyramids-CP is coherent with the actual timing model.**

We use Pyramids-CP to improve the latch placement of an already optimized 130nm commercial ASIC with clock period 2.2ns and 3 million objects. We select the most critical latches that are not optimized, and we use the algorithm from [18] to perform buffering afterward. We compare Pyramids-CP with the Center Of Gravity (COG) method, which is the current and intuitive solution in an industrial physical synthesis flow to improve the latch placements. In COG, the latch is placed to the center of gravity of adjacent pins. The "center" is weighted by the slack of all connected pins. The COG is tested in the same framework as the Pyramids. Table 5 shows a comparison between Pyramids and slack-weighted COG on 10 latches, which are the same latches that are reported in Table 4. On average, the Pyramids-CP improves slack more than 40%, while the COG improves about 16%.

| Pyramids vs. Center-of-gravity | | | | | | |
|---|---|---|---|---|---|---|
| | Pyramids Slack (ps) | | | COG Slack (ps) | | |
| Subcircuit | orig | new | imprv. | orig | new | imprv. |
| latch A0 | -953 | -390 | 563 | -953 | -615 | 338 |
| latch A1 | -897 | 356 | 1253 | -897 | -78 | 819 |
| latch A2 | -706 | 123 | 828 | -706 | 79 | 784 |
| latch A3 | -690 | 395 | 1085 | -690 | -690 | 0 |
| latch A4 | -690 | 173 | 863 | -690 | -690 | 0 |
| latch A5 | -681 | 376 | 1058 | -681 | -681 | 0 |
| latch A6 | -679 | 351 | 1030 | -679 | 209 | 888 |
| latch A7 | -633 | 290 | 923 | -633 | -633 | 0 |
| latch A8 | -614 | 440 | 1054 | -614 | -614 | 0 |
| latch A9 | -610 | 262 | 872 | -610 | 67 | 677 |
| avg | -715 | 238 | 953 | -715 | -365 | 351 |

**Table 5: Comparison of Pyramids-CP with the COG.**

Pyramids-CP formulates a similar problem to that in RUMBLE [8] and is used in place of an LP solver. We compare Pyramids-CP with two open-source LP solvers, the COIN-OR Linear Pro-

| Pyramids-DP Experimental Results | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuits | Original (ps) | | | Pyramids (ps) | | | TAT | Change (%) | | | CPU |
| | $WNS$ | $TNS$ | $WL(10^6)$ | $WNS$ | $TNS$ | $WL(10^6)$ | (ps) | $WNS$ | $TNS$ | $WL$ | $(s)$ |
| s838_1 | -117 | -5415 | 29.3 | -56 | -1117 | 29.4 | 2100 | 52.3 | 79.4 | 0.1 | 2.5 |
| s1196 | -206 | -16691 | 42.0 | -189 | -13733 | 42.4 | 1856 | 8.2 | 17.7 | 1.0 | 1.6 |
| s1238 | -110 | -5388 | 53.3 | -28 | -981 | 53.9 | 1975 | 74.8 | 81.8 | 1.2 | 3.6 |
| s1423 | -232 | -44414 | 44.6 | -96 | -7481 | 45.3 | 4168 | 58.5 | 83.2 | 1.7 | 6.2 |
| s1488 | -110 | -3732 | 58.7 | -91 | -2999 | 58.7 | 1985 | 17.8 | 19.6 | -0.0 | 3.5 |
| s1494 | -118 | -2526 | 61.0 | -17 | -162 | 61.1 | 2122 | 85.9 | 93.6 | 0.2 | 3.2 |
| s15850 | -252 | -27234 | 43.2 | -219 | -21862 | 43.1 | 4532 | 13.0 | 19.7 | -0.3 | 6.2 |
| s9234_1 | -287 | -47904 | 79.4 | -244 | -31104 | 79.8 | 5174 | 15.2 | 35.1 | 0.5 | 9.3 |
| s13207 | -398 | -154110 | 89.7 | -370 | -112690 | 90.6 | 7165 | 7.1 | 26.9 | 0.9 | 33.0 |
| s5378 | -291 | -60104 | 121.7 | -205 | -27543 | 122.6 | 5234 | 29.4 | 54.2 | 0.7 | 11.8 |
| s38584 | -528 | -30212 | 807.2 | -473 | -22912 | 809.2 | 25857 | 10.4 | 24.2 | 0.2 | 107.7 |
| s35932 | -678 | -299320 | 947.4 | -620 | -164599 | 950.5 | 33207 | 8.5 | 45.0 | 0.3 | 137.7 |
| s38417 | -975 | -355126 | 782.0 | -834 | -258295 | 783.1 | 33130 | 14.5 | 27.3 | 0.1 | 76.9 |
| avg | | | | | | | | **30.4**% | **46.7**% | 0.5% | |

**Table 2: Pyramids-DP Results on ISCAS benchmarks.**

gramming (CLP) solver [9] and the GNU Linear Programming Kit (GLPK) LP solver [10]. By using the same LP formulation, the optimal point generated by LP solvers is always in the lower left corner of the optimal region generated by Pyramids-CP. Therefore, there are no significant differences in solution quality. To measure the runtime of all solvers accurately, we save the timing driven placement LP formulations into stand alone problem specifications. One specification file for each problem. To exclude the I/O runtime, for each solver, we run each of the 100 LP problems 10,001 times, then each of the 100 LP problems 1 time. The runtime difference between two runs is the solving time for 100x10000 LP problems. The runtime data is shown in Table 6. Column *PY* shows the runtime of the Pyramids-CP solver. Column $\frac{PY}{CLP}$ shows the speedup of Pyramids-CP over CLP LP solver and column $\frac{PY}{GLPK}$ shows that of Pyramids-CP over GLPK LP solver.

| Exp. | CLP | GLPK | PY | $\frac{PY}{CLP}$ | $\frac{PY}{GLPK}$ |
|---|---|---|---|---|---|
| 100x10000 | 1528.1 (s) | 1838.8(s) | 4.1 (s) | 373 | 448 |

**Table 6: Runtime comparison of Pyramids to CLP and GLPK.**

Pyramids has significant computation advantage over both LP solvers in CLP and GLPK. To find the optimal locations for a cell, LP solvers need to run many iterations to reach an optimal solution, while Pyramids generates the optimal solutions by using a set of closed-form equations. In our experiments, the runtime of Pyramids is hundreds of times faster than that of the LP solvers.

# 6. CONCLUSION

We present Pyramids, an effective and efficient timing-driven placement algorithm. We propose two Pyramids formulations for different wire models, which are suitable for different stages in a physical synthesis flow. Experimental results validate the effectiveness of both formulations. Pyramids-DP improved the timing of a set of circuits, which have been optimized by Cadence SOC encounter. The Pyramids-CP algorithm improved the slack by 40% of cycle time on average for a large commercial ASIC design. Previously, LP-based placement was only feasible for incremental replacement due to runtime limitations. The significant runtime advantage of the Pyramids solver makes the LP-based formulation feasible for use in large-scale timing-driven placement, especially in early stages when timing does not need to be accurately updated after every change.

# 7. REFERENCES


[1] B. Halpin, C. Y. R. Chen, and N. Sehgal, "Timing driven placement using physical net constraints," in *Proc. Design Automation Conf.*, pp. 780–783, 2001.

[2] T. T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. Int. Conf. on Computer Aided Design*, pp. 172–176, 2002.

[3] H. Ren, D. Z. Pan, and D. S. Kung, "Sensitivity guided net weighting for placement driven synthesis.," in *Proc. Int. Symp. on Physical Design*, pp. 10–17, 2004.

[4] W. Choi and K. Bazargan, "Incremental placement for timing optimization," in *Proc. Int. Conf. on Computer Aided Design*, p. 463, 2003.

[5] Q. B. Wang, J. Lillis, and S. Sanyal, "An lp-based methodology for improved timing-driven placement," in *Proc. Asia and South Pacific Design Automation Conf.*, 2005.

[6] A. Chowdhary, K. Rajagopal, S. Venkatesa, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine?," in *Proc. Design Automation Conf.*, pp. 801–806, 2005.

[7] T. Luo, D. Newmark, and D. Z. Pan, "A new LP based incremental timing driven placement for high performance designs," in *Proc. Design Automation Conf.*, 2006.

[8] D. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. Alpert, and I. Markov, "Rumble: An incremental, timing-driven physical -synthesis optimization algorithm," in *Proc. Int. Symp. on Physical Design*, 2008.

[9] COIN-OR linear programming, "https://projects.coin-or.org/clp,"

[10] GNU Linear Programming Kit, "http://www.gnu.org/software/glpk/,"

[11] R. Otten, "Global wires harmful?," in *Proc. Int. Symp. on Physical Design*, pp. 104–109, Apr. 1998.

[12] C. J. Alpert and et.al., "Accurate estimation of global buffer delay within a floorplan," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[13] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integration, the VLSI Journal*, vol. 21, pp. 1–94, 1996.

[14] P. C. P. Saxena, N. Menezes and D. A. Kirkpatrick, "Repeater scaling and its impact on cad," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[15] OpenAccess, "http://openeda.si2.org/,"

[16] Z. Xiu, D. A. Papa, and et.al., "Early research experience with OpenAccess gear: an open source development environment for physical design," in *Proc. Int. Symp. on Physical Design*, 2005.

[17] OAGear:, "http://openedatools.si2.org/oagear/,"

[18] C. J. Alpert and et al., "Fast and flexible buffer trees that navigate the physical layout environment," in *Proc. Design Automation Conf.*, 2004.