An Efficient Surface-Based Low-Power Buffer Insertion Algorithm

Rajeev R. Rao, David Blaauw, Dennis Sylvester, Charles J. Alpert*, Sani Nassif*
Department of EECS, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI
*IBM Corporation, 11400 Burnet Road, Austin, TX
{rrrao, blaauw, dennis}@eecs.umich.edu, *{alpert, nassif}@us.ibm.com

Abstract

Buffer insertion is an important technique used to achieve timing closure in high performance VLSI designs. As the number of buffers in ASIC designs has increased with process scaling, the power consumption of buffers has become a critical concern. In this paper, we present an efficient algorithm that performs van Ginneken style buffer insertion on RC trees and minimizes the total power consumption under a given delay constraint. Our algorithm is based on a formulation that uses a buffer library consisting of continuous buffer sizes. We construct solution candidates in the form of surfaces in the 3-D delay, capacitance and power (DCP) space and show the mechanisms to propagate and merge them in the interconnect tree. Instead of a single minimal power solution, the algorithm produces an entire DCP surface from which a suitable solution point can be selected. We also present a post-processing step where buffers with continuous (non-standard) sizes are snapped to discrete size values corresponding to the buffers in a given library. The proposed algorithm has a worst-case runtime complexity that is polynomial (quadratic) in the number of possible buffer locations. We implemented and tested our proposed algorithm on a number of large benchmark nets and observed that our method produces a speedup in runtime of 5-6X in comparison with previous power aware buffer insertion methods.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms

Algorithms, Design

Keywords

Buffer insertion, physical synthesis, low-power design

1 Introduction

Continued technology scaling has resulted in interconnect delay increasing substantially to significantly influence the total delay of a circuit. Buffer insertion and wire sizing are two well-known techniques that are used to reduce the wire delays of critical signal nets in a circuit. The delay of a wire is proportional to the product of the resistance and capacitance of the wire. Since both the capacitance and resistance are proportional to the length of the wire, the wire delay has a quadratic dependence on the wire length. Buffer insertion divides the wire into a number of smaller segments and linearizes the wire delay with respect to the total length of the wire [1]. Furthermore, the inserted buffers have the additional benefit of decoupling large loads from the critical path of the circuit.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISPD'05, April 3–6, 2005, San Francisco, California, USA.
Copyright ACM 1-59593-021-3/05/0004...$5.00.

The buffer insertion problem takes as input a routing tree with a single source and multiple sinks and seeks to find the set of edges in the tree that need to be buffered so that the delay of the circuit is minimized. In [2], van Ginneken presented a dynamic programming algorithm for optimal buffer insertion that has been the basis for most of the later work in this area. If the buffer library consists of a single buffer, the van Ginneken algorithm has a worst-case complexity that is quadratic in the number of possible choices for buffer locations. The extension of the algorithm to minimize the number of buffers was shown to have non-polynomial complexity.

In [3], the authors proposed an approach to perform multiple buffer insertion on a single edge and extended the approach in [4] to perform noise optimization. In [5], the authors use a quadratic programming approach to present an optimal polynomial runtime algorithm to solve the buffer insertion problem. The authors use the method of Lagrangian relaxation in [6] to develop exact algorithms for gate and wire sizing. A clock-tuning algorithm that performs zero skew clock-tree optimization with buffer insertion and wire sizing is proposed in [7]. In [8], the authors present a composite approach to perform simultaneous buffer insertion and driver sizing. All these approaches assume a given routing tree topology. On the other hand, approaches such as [9][10][11] perform simultaneous optimization of the routing tree and buffer insertion. These approaches utilize various techniques, two examples being extended dynamic programming and stochastic search. The authors in [12] provide exact solutions for power-optimal buffer insertion for single interconnect nets while in [13] the authors use the Nash equilibrium function to reduce the switched capacitance in gate level circuits. Recently, the authors in [14] proposed a novel approach to reduce the complexity of the basic buffer insertion algorithm from $O(n^2)$ to $O(n \log n)$.

It has been widely reported that the number of buffers in modern chip designs is rapidly increasing with process scaling [15]. Currently, 50% of the total leakage of a chip is due to buffer devices [16]. Assuming the continuation of current scaling trends and design styles it has been estimated that at the 32nm technology node 70% of the cell count will be due to buffers and repeaters [17]. Since buffer devices are not influenced by the stack effect [18], they contribute strongly towards the leakage power consumption of any design. As a result, buffers now constitute a substantial portion of the total power dissipation of a chip, making low-power buffer insertion techniques critical. To this end, Lillis et.al., [19] proposed an extension of the van Ginneken's algorithm that performs power minimization under a given timing constraint. Their power optimization algorithm targets the dynamic power component and has polynomial time complexity under the assumption that the capacitive values can be mapped to a polynomially bounded integer domain. Under the same assumption, considering a buffer library with multiple buffers adds a quadratic multiplicative factor to the complexity of the algorithm. However, for realistic buffer trees it is likely that wire capacitances take on arbitrary values and hence the runtime complexity cannot be assumed polynomially bounded. Similarly, the runtime complexity with respect to multiple buffer sizes under arbitrary wire capacitance becomes quadratic.

The runtime complexity of existing buffer power optimization algorithms is a severely limiting factor for substantial interconnect trees. This is especially true with high-end standard cell libraries, where the number of available buffers can be very large. In this paper, we therefore propose an alternate approach that solves the problem in the continuous domain and then discretizes the obtained solution to the available discrete cell sizes in the standard cell library. Since the number of available options is large, the final discrete solution is close to the continuous solution and thus such an approach can be expected to be near optimal. A key advantage of the approach is that the runtime complexity can be shown to be quadratic with the number of buffer locations making it possible to optimize large buffer networks for power. Continuous wire sizing [20][21][22] and continuous gate sizing [23][24] algorithms proposed previously have aimed towards optimizing the delay component of timing critical paths in a general circuit. In this paper, we use the concept of continuously sized buffer libraries to perform power optimal buffer insertion on any given net in the circuit.

The proposed approach is based on the propagation of solution surfaces in the delay, capacitance and power (DCP) space through the routing tree instead of *n*-tuple solution points, as is done in the standard Van Ginneken's algorithm. We show the methods by which the DCP surfaces are merged and propagated both with and without buffer insertion at a particular location in the tree. At the end of the initial backward bottom-up phase we provide the user with the optimal surface (in the DCP space) for the source pin of the net. The user then chooses any solution point on this surface based on the desired delay, capacitance and power values. The subsequent top-down phase then determines the buffer locations and the required continuous buffer sizes corresponding to this solution point. An additional post-processing step is utilized to snap back the continuous sizes generated by the bottom-up phase to the discrete sizes available in standard cell library.

The remainder of the paper is organized as follows: In Section 2 we present the delay models that we use for interconnects and buffers. In Section 3 we give a detailed overview of the dynamic programming approach and formulate the problem that we solve in this paper. In Section 4 we present our new approach and highlight the important aspects related to it. In Section 5 we present results from running our algorithm on a standard benchmark set and also compare our results with previous approaches. Finally, we conclude the paper in Section 6.

2 Delay Models

We assume that the routing tree topology is fixed or an initial Steiner estimation is available for a given net. A routing tree $T = (V,E)$ consists of $(n-1)$ wires E and a set of n nodes $V = \{\{so\} \cup SI \cup IN\}$ where so is the unique source node, SI is the set of sink nodes and IN is the set of internal nodes. A wire $e \varepsilon E$ is an ordered pair of nodes (u,v) where the signal propagates from u to v. Without loss of generality, we assume that the tree is binary so that each node can have at most two (left and right) children. The path from node u to v is denoted by $path(u,v)$. The lumped capacitance and resistance of wire e are denoted by R_e, C_e. We are also given a buffer library B where each buffer can take on any size from a given range of sizes. A buffer insertion solution is a mapping $M{:}IN \rightarrow B \cup \{\bar{b}\}$ which either assigns a buffer (of one particular size) or no buffer (indicated here as $\bar{b}$) to each internal node of T. For each $v \varepsilon V$, let $T(v) = (\{v\} \cup SI_{T(v)} \cup IN_{T(v)}, E_{T(v)})$ be the maximal subtree of T (rooted at v) that does not contain any internal buffers. If v is a sink then $T(v)$ contains only one node.

We adopt the Elmore delay model [25] as in the previously published works ([2][3][4][9]) for interconnect (wire) delay and the standard RC (linear) delay model for buffers. For a given routing tree, the root-sink delay is composed of delays due to interconnect (wires) and delays due to buffers. For a subtree rooted at v, the capacitive load seen at node v is the total lumped capacitance $C_{T(v)}$ given by:

$$C_{T(v)} = \sum_{k \in SI_{T(v)}} C_k + \sum_{e \in E_{T(v)}} C_e \qquad \text{(EQ 1)}$$

The Elmore delay for the wire $e = (u,v)$ is given by:

$$Delay(e) = R_e\left(\frac{1}{2}C_e + C_{T(v)}\right) \qquad \text{(EQ 2)}$$

Suppose (d_b, c_b) are the delay and input capacitance associated with a buffer. For the moment we assume that the value of d_b includes information about both the intrinsic delay through the buffer and also the load capacitance that the buffer needs to drive. When a buffer is inserted at the end of a branch in a subtree, the delay through the branch increases by amount d_b while the load capacitance seen by the root of the subtree is just c_b. This is due to the so-called *isolation property* of buffers. Since the buffer decouples its ancestors (parent branches) from its descendants (children subtrees), the capacitive load seen by the ancestors is exactly equal to the input capacitance of the buffer. In general interconnect trees, since a downstream wire contributes a significant amount to the overall output load of any buffer, the impact of the Miller effect is negligible.

The total delay from $v \varepsilon V$ to $si \varepsilon SI$ is given by:

$$Delay(v, si) = \sum_{e = (u,w) \in path(v, si)} Delay(e) + Delay(u) \qquad \text{(EQ 3)}$$

Most previous works ([3][4][19]) use the maximization of required arrival time (*RAT*) at the root as their timing metric. However, for our algorithm, we instead use an equivalent but different metric known as subtree delay (*SD*). We assume that in the input RC tree network, each sink *si* has an $RAT(si)$ associated with it. Among, all such sink *RAT* values we extract the maximum value $RAT_{max}(si)$ and set the subtree delay values for all the sinks as $SD(si) = (RAT_{max}(si) - RAT(si)) \geq 0$. As we traverse up the tree in a bottom-up fashion from the sinks to the source pin, the delays due to interconnects and buffers are accumulated. As we observe in Section 4, this accumulation property is essential for the efficient generation of curves and surfaces for our algorithm. Once the backward traversal is completed the subtree delay for the source $SD(so)$ is generated. The given tree network meets timing if $SD(so) \geq RAT_{max}(si)$. Note that this is exactly equivalent to the *RAT* formulation except for the fact that we have taken the *time*=0 point at a sink instead of the source.

3 Previous Approaches

The well-known van Ginneken algorithm [2] for buffer insertion is based on the dynamic programming approach and is briefly reviewed in this section. The buffer placement algorithm computes the delay and load values using recursive formulae and constructs possible candidate solutions for each node in the tree. The algorithm proceeds in two separate phases. In the backward phase, the delay paths are traversed from all sinks to the source and at each node a set of candidate solutions is generated. In the forward solution phase, the candidate with the best delay value is chosen and the tree structure with the appropriate buffer locations is reconstructed. The pseudo-code for the backward pass (given by function BOTTOM_UP) is given in Figure 1. In this figure, SD_v is the subtree delay value at node v while C_v is the capacitive load that node v presents upward to its ancestor nodes.

The VG algorithm consists of four major steps (Lines 5-8). First, once the candidate set for the left and right children has been determined, a composite candidate needs to be constructed using a merge operation. For a node that is driving two separate child branches, the capacitive load will be the sum of the two loads while the delay value will be the maximum of the two *SD* values. Next, among the

```
Function BOTTOM_UP (v)
1. If v ε sink { return (C_v, SD_v) } Else
2. /* compute options for subtrees */
3. BOTTOM_UP(left(v))  /* Create cnds for left subtree */
4. BOTTOM_UP(right(v))  /* Create cnds for right subtree */
5. Join pairs of subtrees by a merge operation
6. Find best cnd among merged cnds to add a buffer
7. Add parent wire to both types of cnds
8. Prune inferior cnds from set of cnds
9. Store cnd list for node v and return
```

Figure 1. Pseudo-code for the function BOTTOM_UP

merged solutions, the candidate that corresponds to the best buffer insertion solution is extracted. Note that exactly one solution among the merged candidates is chosen for the buffer insertion operation. For all the buffered and non-buffered solutions, the delay and load values are updated using the information of the driving parent wire. Finally, an appropriate pruning mechanism is used to remove the provably sub-optimal set of candidates. The basic VG algorithm has time complexity O(n^2) where n = number of buffer locations. The pruning mechanism is useful to remove unneeded candidate solutions and helps reduce the sizes of the candidate sets. Furthermore, if the candidate solutions for each node are stored in an appropriate manner (for instance, ordered by decreasing *SD* values) then the number of solutions from the merge operation will be linear in the number of left/right children candidates.

In [19], Lillis et. al. proposed a number of extensions to the previous algorithm. A complete buffer library can be used to select the appropriate set of buffers that are to be inserted. For a buffer library with $|B|$ buffers, the time complexity of the basic buffer insertion algorithm is O($n^2|B|^2$). Wire sizing can be integrated by viewing each possible wire size as a choice from a wire library. Signal slew can be included into the simple gate delay model at the cost of increased time complexity. An important contribution of this work is the inclusion of dynamic power consumption into the buffer insertion model. The authors present a method to minimize the capacitive load of an RC tree network subject to the condition that the circuit meets the required timing constraint. To accomplish this feature, the authors include a "power" number as part of the candidate solutions. The presence of this power number is accounted for by appropriately modifying the four major steps in Figure 1.

In general, the addition of extra objectives into the dynamic programming formulation drastically increases the worst-case time complexity. When a "cost" function, such as the minimization of the number of inserted buffers is included, the time complexity is no longer polynomial. For the case when a power objective is included, Lillis et. al. assume that the capacitive values can be mapped to polynomially-bounded integers and show that such a formulation still results in polynomial (but worse than quadratic) time complexity. However, the authors only consider the dynamic power dissipation in their analysis. In modern technologies, the increased impact of leakage skews the value for total power, thereby invalidating the assumption that power numbers can be mapped into a set of polynomially-bounded integers. Hence, directly including the total power value as an extra field in the candidate set will result in exponential worst-case time complexity.

Among the previously proposed buffer insertion algorithms, a few methods can be extended to handle the power consumption aspect of interconnect trees. In the quadratic programming approach given in [5], the authors equate wire area (and interconnect capacitance) with the power consumption of the net. They solve the case for area minimization with bounded delay by applying Lagrangian relaxation [6] to handle the constraints on arrival time. In the clock-tuning algorithm presented in [7], the authors use the capacitance shielded by buffers as a metric for determining the power of a signal net. However, such approaches are inherently problematic because they ignore the increasingly relevant leakage power aspect of buffers. The inclusion of buffer power into the quadratic programming formulation would make the resulting mathematical program non-quadratic [5] and also violate the requirements for the Lagrangian relaxation [6] method. For the clock-tuning algorithm, the augmentation of buffer leakage power into the problem framework would significantly increase the required number of sampling steps for the three-dimensional DC regions [7] thereby worsening the complexity of the algorithm.

Instead of a buffer library with a set of discrete (and possibly unrelated) buffers, we assume that our library consists of buffers that are continuously sized. The concept of continuously sized cell libraries has been prevalent [26][27] and CAD techniques to leverage optimization algorithms using such libraries have been developed [22][28][29]. For such a continuous library, a straightforward application of the VG algorithm is unsuitable since approximating the continuous library with a fine grain discretization leads to large run times, especially if power is considered to be a part of the candidate solution set. In this paper, we therefore propose a novel continuous buffer insertion algorithm that includes power minimization as an objective. As detailed in the next section, our approach generates a complete 3-D surface of the delay-load-power trade-off for each node in the circuit. The proposed approach has two advantages. First, it allows the user to pick any desired trade-off point from a delay-load-power (DCP) surface. This allows the user to easily explore trade-offs between different variables. From any trade-off point, we generate the complete RC tree network with optimally sized buffers at the appropriate locations. We also provide a method to snap the buffer sizes back to a discrete library if a continuous library is not available. Second, we prove in the next section that our algorithm has worst-case time complexity of O(n^2). Therefore, instead of a continuous buffer library we use one that is composed of a set of buffers with discrete sizes resulting in the proposed algorithm having a runtime advantage.

4 Proposed Algorithm

We use the same dynamic programming framework of van Ginneken's algorithm. We use a backward bottom-up pass to generate candidate solutions for each node and a forward top-down solution pass to extract the required solution. Since the buffer library has continuously sized buffers, it is not possible to iterate through these sizes for each node. As we discuss in Section 4.1, we characterize the entire buffer library using empirical equations that relate sizes with the relevant buffer parameters. From this formulation, we see that when a buffer is included as a candidate solution a range of possible delay numbers corresponding to an optimal power-delay trade-off curve emerges. Furthermore, as this curve is propagated back to the next buffer location, a complete 3-D surface is created corresponding to power, delay and load trade-offs. We define three types of candidates:

- Point - A point candidate is a 3-tuple of (capacitance, delay, power). The Lillis et. al. extension [19] to the standard VG algorithm contains only the 3-tuples as the candidate solutions. Every sink *si* consists of a single point candidate (*C*(*si*), *SD*(*si*), *P*(*si*)) where *C*(*si*) is the output load at the sink and *P*(*si*) is the power dissipated by that output load. For an internal node *v* in the tree, a point candidate corresponds to the solution when there are no buffers in the subtree rooted at *v*.
- Curve - A curve candidate is a pair of (Capacitance, Delay) and (Power, Delay) curves for a given range of delays. A curve candidate corresponds to the solution when there is exactly one buffer in the subtree rooted at a node.
- Surface - A surface candidate consists of a set of capacitance values and power-delay curves associated with each such load value. A surface candidate corresponds to the solution when there are two or more buffers in the subtree rooted at a node.

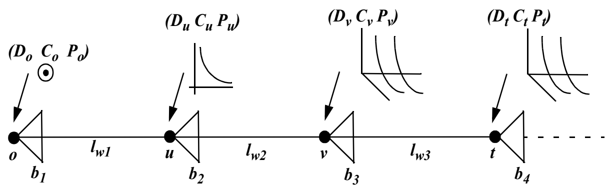

Figure 2. An interconnect line with four inserted buffers. Node *o* has a point candidate, node *u* has a curve candidate, nodes *v*, *t* have surface candidates

A detailed explanation about these candidates as well as their algorithmic operations (merging, pruning) is presented in Section 4.2 and Section 4.3. In Section 4.4 we discuss the operation of the snapping mechanism once the candidate solution is generated for the source node and in Section 4.5 we provide the complexity analysis for our algorithm.

4.1 Buffer Library Characterization

We consider a buffer library consisting of a set of continuously sized buffers. For a range of fine-grained sizes, the library potentially contains a large number of buffers with different sets of drive strengths and input capacitances. We set the P/N ratio of the buffers to be 2. Let S be the sizing factor corresponding to the integer multiples of the transistor widths.

For each buffer in the given library, we empirically fit a set of polynomial functions to express delay (d_b), input capacitance (c_b), and leakage current (l_b) as a function of size (S) and output load (C_{out}):

$$d_b = d_0 + d_1 \cdot \left(\frac{C_{out}}{S}\right) \qquad c_b = c_0 + c_1 \cdot S \qquad l_b = l_0 + l_1 \cdot S \qquad \text{(EQ 4)}$$

Here d_0, d_1, c_0, c_1, l_0 and l_1 are fitting constants. These equations are very similar to the linear buffer delay model given in [19]. Note that while buffer delay is dependent on both its size and output load, the input capacitance and leakage values are simple linear functions of the sizing factor S. EQ4 enables us to combine a small number of discrete buffer sizes to approximate the ideal of continuous buffer sizing.

4.2 Creation of a curve candidate

As we mentioned previously, there are three types of candidates - Points, Curves and Surfaces. We first present the basic configurations in which Curve and Surface candidates are generated.

Consider the circuit in Figure 2. A subtree rooted at node t has three wires of length l_{w1}, l_{w2} and l_{w3}. The buffer locations are indicated in the figure. The sink node in the subtree has a Point candidate of the form (D_o, C_o, P_o) and we need to generate the candidate solution at t. Let r_w and c_w represent the resistance and capacitance per unit length of a wire. We begin the backward pass by first writing the equations for delay (D_u), capacitance (C_u) and power (P_u) seen by the subtree rooted at the intermediate node u.:

$$D_u = D_o + d_{b1} + d_{wire1} = D_o + \left[d_0 + d_1 \cdot \left(\frac{C_0}{S}\right)\right] + \frac{1}{2} r_w c_w l_{w1}^2 + r_w l_{w1} c_{b1}$$
$$C_u = c_{b1} + c_{wire1} = [c_0 + c_1 \cdot S] + c_w l_{w1} \qquad \text{(EQ 5)}$$
$$P_u = P_o + l_{b1} + p_{wire1} = P_o + k_L[l_0 + l_1 \cdot S] + k_D(C_u)$$

Here k_L and k_D are proportionality constants used to express total power as a scaled version of the leakage current and capacitive load. For each parameter we express the contributions of the point candidate, the buffer and the wire separately. As highlighted in Section 2, the isolation property decouples the ancestors of the buffer from its descendants. Thus, the expression for C_u does not depend on C_o. For different values of S, we tabulate the values for D_u, C_u and P_u. We then empirically fit these values using the following functions:

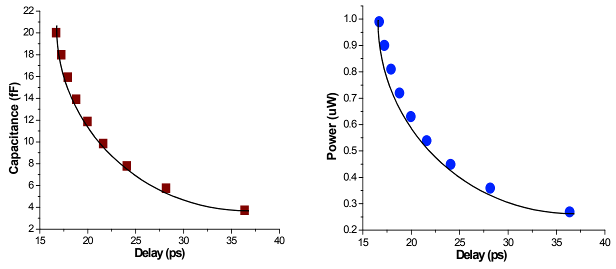

Figure 3. Generation of curve candidates for node *u*. Sink candidate at node *o* is of the form (D_0, C_0, P_0 = 0.0, 10.0, 0.144)

$$C_u = g_0 + (g_1 \cdot D_u) + (g_2 \cdot D_u^2) \qquad P_u = k_0 + (k_1 \cdot D_u) + (k_2 \cdot D_u^2) \qquad \text{(EQ 6)}$$

The g_i's and k_i's are fitting constants. We have utilized quadratic equations for our empirical model because (1) they provide an excellent fit for the given data and (2) the fitting coefficients can be efficiently determined using multilinear fitting models. EQ6 expresses C_u and P_u in terms of D_u using a quadratic equation. Note that a subtree delay (SD) based formulation ensures that the delay values always remain positive and the shapes of the reciprocal curves do not change. (In EQ5, with a *RAT* based formulation, the d_b term would need to be subtracted and hence the shape of the *C/D* and *P/D* curves would be altered). In addition to the fitting constants, we store the range for D_u = [D_{umin}, D_{umax}] to fully describe the curve candidate that is generated at node u. In Figure 3, we plot a sample curve candidate for a given input point candidate. The corresponding sink candidate at node o is given by (D_o, C_o, P_o) = (0.0, 10.0, 0.144). From the plot, we clearly see that a quadratic polynomial fit as given by EQ6 is sufficient to capture the relationships between C_u/D_u and P_u/D_u.

4.3 Creation of a surface candidate

We again consider the circuit from Figure 2. A buffer is inserted at the end of l_{w2}. We can write the equations for delay (D_v), capacitance (C_v) and power (P_v) seen by the subtree rooted at node v.

$$D_v = D_u + d_{b2} + D_{wire2} = D_u + \left[d_0 + d_1 \cdot \left(\frac{C_u}{S}\right)\right] + \frac{1}{2} r_w c_w l_{w2}^2 + r_w l_{w2} c_{b2}$$
$$C_v = c_{b2} + c_{wire2} = [c_0 + c_1 \cdot S] + c_w l_{w2} \qquad \text{(EQ 7)}$$
$$P_v = P_u + l_{b2} + p_{wire2} = P_u + k_L[l_0 + l_1 \cdot S] + k_D(C_v)$$

This equation is similar to EQ5. Due to the isolation property, the capacitance seen at node v is dependent only on the size of the newly inserted buffer and the parent wire length. However, we see from EQ7 that the delay and power values at v are dependent on the sizes of both the buffers. Considering the 3-tuple (D_v, C_v, P_v) in the 3-D DCP space, we see that for a given value of c_{b2} (i.e., a given size for the second buffer) the value of C_v is fixed while D_v, P_v assume values based on the values of the size of the first buffer. Hence, we see that for a given plane on the C-axis in the DCP space, there exists one P/D curve. We use the term C-plane for such planes and illustrate the resultant set of planes in Figure 4. For each C-plane, we refit the P/D curve using a quadratic equation (EQ6) and obtain a new set (k_i's) of fitting constants.

$$P_{v|C_v} = k_0 + (k_1 \cdot D_v) + (k_2 \cdot D_v^2) \qquad \text{(EQ 8)}$$

In EQ8, the term $P_{v|Cv}$ is used to represent the power-delay curve that is constructed for a particular value C_v corresponding to a specific C-plane. For the general data structure of a C-plane, in addition to the fitting constants k_i's, we store the delay range [D_{vmin}, D_{vmax}]

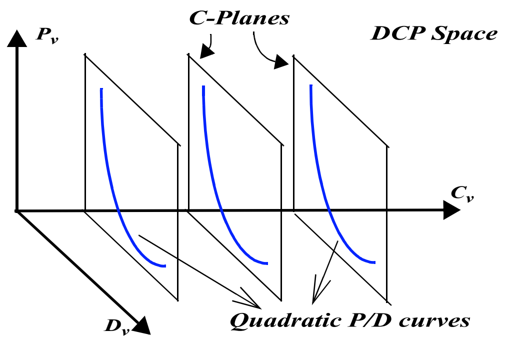

Figure 4. Generation of *C*-Planes in the DCP space. The complete set of *C*-planes and the quadratic *P/D* curves associated with each such plane describes a surface candidate

corresponding to only those delay values that are achievable for that particular *C*-plane. We repeat this computation for different values of the second buffer size and generate different *C*-planes. The complete set of *C*-planes with the associated family of *P/D* curves describe a general 3-D surface in the DCP space.

Continuing in the same vein, suppose there was a third buffer inserted in the circuit given in Figure 2 at node *t*. We can rewrite the new *D*, *C*, *P* equations for this candidate in a manner similar to EQ7. However, for this new candidate, while the capacitance value will depend only on the size of the third buffer, the delay and power values will be dependent on the delay/power values of the points on the different *C*-planes of the surface candidate at node *v*. Hence, in the new *C*-plane, we obtain a set of *P/D* curves instead of a single curve as illustrated in Figure 5. Given a set of such *P/D* curves, the power optimal solution occurs when we pick the minimum power solution for any given delay and capacitance value. Such a minimum power solution is obtained by taking the lower envelope of the set of *P/D* curves for each *C*-plane. The lower envelope is illustrated by the solid line in Figure 5. This curve is empirically fit using a quadratic equation similar to EQ6. For each buffer size corresponding to the third buffer, we create a new *C*-plane and a new lower envelope *P/D* curve. The complete set of *C*-planes with the associated *P/D* curves describes a general 3-D surface in the DCP space for node *t*. Thus, we have shown that given a node with a surface candidate, adding another buffer to it will result in yet another surface candidate at the new node. Figure 4 shows the general structure of a surface candidate in the DCP space. For the sake of simplicity we have drawn all the *C*-Planes to be composed of identical *P/D* curves but this will certainly be different in the general case.

4.3.1 Merging candidates

In Figure 2 we only focussed on the simple case when a node has a single child node. For general binary trees where nodes have left and right children, we need to perform a merge operation and extract

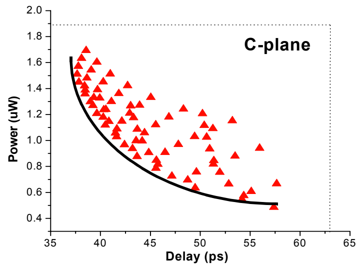

Figure 5. Developing the *P/D* curve for a single *C*-Plane. For multiple *P/D* curves, the lower envelope gives the minimum power solution for given delay/power values

the best candidate to which a buffer can be added. In our analysis we only consider nets that are modeled as binary trees. A non-binary tree can be trivially converted to a binary tree by the insertion of dummy nodes and zero length wires. We assume that our wire library consists of a single wire. We further assume that all the buffers in the library are non-inverting. We can easily extend our method to handle the case with both inverting and non-inverting buffers by splitting [19] the total candidate solution sets at each node into a pair of disjoint subsets, each corresponding to one type of buffer.

We now illustrate the merging operation for the case when the left and right children are both surfaces since this situation represents the worst case scenario for the algorithm in terms of time complexity. The other cases for merging pairs of curves or pairs of different types of candidates can be handled in a similar manner. In our algorithm, we limit the number of candidates per node to two - one corresponding to a buffered solution and the other corresponding to a non-buffered solution. Note that the number of solutions that are maintained at each node does not affect the optimality in a direct manner. A larger number of solutions will improve the accuracy of the fit for the power delay curves at the parent node. We observed that maintaining just two solutions provided sufficient accuracy in the fitting process while enabling efficient analysis.

Given two candidates from each of the children of a node, the merging of these candidates for the left and right children into a single candidate is accomplished in two steps. First, we join each possible pair of left and right candidates resulting in four candidates, and then reduce these four candidates into a single candidate. While joining a pair of candidates from the left and right children it is important to compare equal delay points since any combination of unequal delay values would be clearly suboptimal. We therefore enumerate a new set of delay values and visit each of the *C*-planes in the left and right candidates. For any given *C*-plane, if the delay value is part of the delay range associated with the *C*-plane's *P/D* curve, then we use the fitting function (EQ6) to calculate the power value. If it is not part of the range, then by definition, that set of delay points does not influence the timing requirements of that portion of the tree network. Hence, we pick the lowest power value (i.e., the power number for the slowest delay value) for that set of points.

This operation results in left and right *P/C* curves corresponding to each such delay value. We then discretize these *P/C* curves into *z* points and form all possible combinations of left and right *P/C* points. This creates a new set of z^2 *P/C* points from which we obtain the optimal *P/C* curve by taking the lower envelope of this set of *P/C* points. In this process, we eliminate the set of points that have the same *C* value but higher *P* values, as they are obviously dominated by the points on the lower envelope curve. After a combined *P/C* curve is generated for each delay value, we translate the *P/C* curves for fixed delay points into a set of *P/D* curves for fixed *C* values. For each such fixed *C* value, we create a new *C*-plane and refit the *P/D* curves to obtain a new set of fitting parameters. A complete set of such *C*-planes generates a new surface candidate.

This merging operation results in four separate surfaces corresponding to combinations of buffered and non-buffered solutions for the left and right children. We then execute an extra combination operation in order to select the minimum power value corresponding to each *(D, C)* value. This is done by comparing a set of four *P/D* curves in the same *C*-plane. Typically, the four solutions have different sets of capacitance values for the *C*-planes. Hence, we may be required to map *P/D* curves from one *C*-plane to another. This mapping is accomplished using a simple linear interpolation method. This combination operation produces a composite surface from which we construct the buffered candidate as illustrated in Section 4.3. While it is possible that there is some amount of fitting error introduced due to the merge operation, we find in practice that this error due to the refitting of quadratic *P/D* curves is negligible. Even for the benchmark nets with large source-sink depths, the accumulation of fitting errors during the bottom-up traversal was minimal and there was only a slight deviation from the optimal solution points.

Once the backward pass is complete, a pair of candidate solutions is created for all nodes. For the sake of simplicity, we assume that there is no driver gate present at the source. (However, including such a driver gate is trivial since it only displaces the candidate solutions for the source in the DCP space). Given the buffered and non-buffered candidate surfaces at the source node, we pick an appropriate point in the DCP space as our sample solution point. Using this solution point, we perform a forward pass through the entire tree to reconstruct the tree structure that produced that sample solution point. During this reconstruction phase, we determine the locations as well as the exact sizes of the buffers that need to be inserted. Finally, we use the snapping mechanism described in the next section to ensure that only the discrete sizes that are part of the buffer library are present in the final solution.

4.4 Snapping to Required Size

If a continuous buffer library is not available, the final solution must be snapped to discrete buffer sizes. Hence, we snap buffers with non-integer sizes to have the nearest integer size. Such a modification obviously introduces discrepancies in all three components (delay, load, power) of a solution point. In particular, changing the buffer size alters the leakage and input capacitance directly while the delay values are affected as a result of the parent node of the modified buffer encountering a different downstream load value. We first identify the buffers that need to be snapped and then recompute the RC tree delay and load values in a bottom-up fashion. In our implementation, we also incorporate local adjustments to account for the small amounts of delay that may be lost due to the snapping method. Since each candidate consists of entire sets of power-delay curves, the local corrections can be accomplished by shifting the delay values on the *P/D* curves appropriately as we traverse from the root of the tree to the sink node. The resultant solution after the snapping operation is therefore ensured to be very close to the sample solution point that we chose originally and, consequently, we avoid an accumulation of rounding errors. Note that after the final solution is generated, the snap correction can be applied in just a single pass through the RC tree network.

4.5 Run Time Analysis

The algorithmic framework is the same as the basic van Ginneken dynamic programming algorithm. The main issue that needs to be considered is the number of discretizations and its effect on runtime. For a value of *z* as defined in the previous section, we see that in the worst case, the merge operation needs to compare left and right surfaces that are both discretized using *z* *C*-planes. For the sake of simplicity we have set *z* to be equal to the number of discrete size choices in the buffer library. Thus, for each surface candidate, the number of *C*-planes is exactly equal to the number of discrete size choices. The complexity of the four merge operations is $O(z.z) = O(z^2)$. Adding a buffer candidate is achieved by visiting all *C*-planes for the "best" candidate surface and creating *P/D* curves for various buffer sizes in the new *C*-planes. This operation can also be done in $O(z^2)$. During merging, the lower envelope operation locally prunes the sub-optimal solutions. Hence additional pruning steps are not required. Finally, we utilize fitting functions from a standard math library and since each fit is done using *z* discretizations, the amortized total time for this fitting is of the order $O(n.z)$. Thus time complexity of the algorithm is $O(n^2+n.z^2+n.z^2) = O(n^2)$ since typically *z* is small constant number.

5 Results

We implemented the proposed algorithm on the framework of a standard van Ginneken algorithm. We used C++/STL for our implementation and the runtimes reported are from a Pentium IV 2.4 GHz machine with 1GB RAM running Linux. We have used the GNU scientific libraries (GSL) [30] for the multilinear fitting models. Our set of benchmarks come from the C-tree based nets presented in [31] and available at [32]. In Table 1, for each of the benchmark nets, we report the number of sinks and the number of edges with non-zero wire length (these are the non-dummy wires in the net).

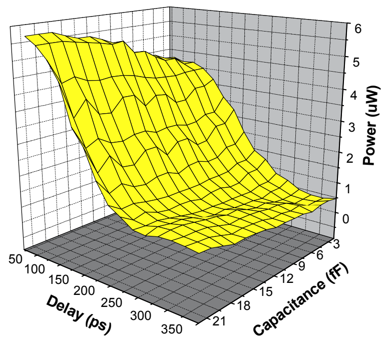

Figure 6. Sample candidate 3-D Surface at the source node. For a given capacitance, there is a quadratic relationship between delay and power

For our algorithm, we assume that a Steiner tree based estimation is available for the given net. The continuous sized library is constructed by considering a large set of buffers available from a standard TSMC 0.13μm standard cell library. We first use SPICE to characterize this library as described in Section 4.1 and obtain the fitting constants described in EQ6. For our experiments, we assumed that the number of discrete buffer choices in the library $z = 9$. The values for the resistance and capacitance per unit length are technology specific and are obtained from ITRS [33]. In Figure 6, we first present a sample 3-D surface for the candidate at the source node for a given benchmark. For a given capacitance value (i.e., a given *C*-plane) we see the quadratic curve between power and delay.

In Table 1 we show a capacitance, delay, and power trade-off point selected from the 3-D trade-off surface for each benchmark. From the pair of candidate choices (buffered and non-buffered) at the source node, we first pick a sample 3-tuple point (Cap (fF), Del (ps), Pow (μW)) on the DCP surface corresponding to the pre-snap solution. We execute the forward solution pass to determine the continuous buffer sizes that will result in the required solution point. We then utilize the snap correction mechanism and determine the appropriate integer buffer sizes that will produce the post-snap solution that is as close as possible to the original pre-snap solution. From the table we see that for all the nets, there is a noticeable difference in the cap values between the pre-snap and post-snap solutions. This is to be expected due to the fact that the capacitance value at the source pin can only attain a fixed set of cap values. When the chosen sample point does not match the cap value of a discrete buffer size, the snapping mechanism forces the first buffer to be of a particular integer size. For instance, both mcu1s9 and n8692 have the same post-snap cap value of 13.9 since that buffer size is the one closest to the initial pre-snap cap value of 14.8 and 14.2. On the other hand, we find that there is only a minor difference between the delay and power numbers. Since we repeatedly perform local adjustments for delay as we progressively snap buffers along the tree, a sufficient tree depth ensures that the post-snap solution produces delay and power values that are close to the original sample solution point. We also provide the total number of inserted buffers for each benchmark net.

We compare our algorithm against the power related buffer insertion algorithm presented in Lillis et. al. [19]. We implemented the Lillis algorithm and tested both methods against the same set of benchmarks. In particular, we compare against their method of buffer insertion for power minimization for the case with $|B| \geq 1$ and

Net	# sinks	# edges	Pre-Snap			Post-Snap			# buffers
			Cap	Del	Pow	Cap	Del	Pow	
mcu1s9	24	30	14.8	276.1	9.9	13.9	276.2	10.0	9
n8692	28	29	14.2	315.5	11.9	13.9	315.9	11.9	11
pointer3	25	33	16.7	311.6	8.9	15.9	311.9	8.8	9
n313	23	31	9.7	442.5	7.8	9.8	441.5	7.7	8
n18905	33	43	7.4	206.8	16.4	7.5	207.1	16.1	16
n7866	34	52	10.8	182.5	16.9	9.8	182.1	17.3	17
n8702	46	60	8.5	196.7	16.9	7.5	196.3	17.1	22
netbig4	79	81	17.3	462.8	26.8	18.0	463.8	26.6	37
netbig3	64	67	17.2	534.0	25.7	18.0	533.2	25.8	30
netbig2	74	79	14.2	531.5	28.9	13.9	533.5	28.8	34
netbig1	91	102	11.7	445.6	22.2	11.9	448.5	22.3	43

Table 1. Comparison of pre- and post-snap solution points. There are differences in the capacitance values but the delay and power numbers are nearly identical

Net	Lillis algorithm [19]				Our algorithm			
	Cap	Del	Pow	Runtime	Cap	Del	Pow	Runtime
mcu1s9	15.2	128.2	8.5	0.34	16.0	127.3	8.6	0.13
n8692	16.9	169.1	8.7	0.37	16.0	168.0	8.6	0.13
pointer3	15.2	154.8	10.6	0.42	16.0	154.9	10.5	0.14
n313	14.9	163.7	10.9	0.43	13.9	162.6	10.8	0.15
n18905	7.6	204.4	16.8	0.69	7.8	204.5	17.0	0.18
n7866	10.2	185.3	17.1	0.82	9.8	185.7	17.3	0.20
n8702	9.0	177.8	17.2	1.01	9.8	178.1	17.3	0.22
netbig4	8.2	212.9	29.6	2.06	7.8	212.7	29.8	0.46
netbig3	11.2	223.8	28.2	2.44	11.9	224.9	28.3	0.43
netbig2	19.0	188.2	35.6	2.80	18.0	189.1	35.6	0.52
netbig1	13.2	215.7	32.7	3.03	13.9	216.8	32.7	0.53

Table 2. Comparison of Lillis et. al. [19] vs. our algorithm. There are differences in the capacitance values but the delay and power numbers are the same. Runtime improvement is about 5-6X

$|W| = 1$ (multiple buffers, single wire). Additionally, for the Lillis algorithm, instead of using a capacitance value to represent the power of the signal net (as originally given in the paper), we included the total power value (inclusive of leakage) as part of the candidate solution. In Table 2 we list the 3-tuples of (Cap(fF), Del(ps), Pow(μW)) as well as the Runtime (seconds) for both methods. We first perform the backward pass for both methods and generate the candidates at the source node. We then pick a sample solution point generated by the Lillis algorithm, execute the forward solution pass and finally apply the snap correction mechanism to obtain the buffer solution with discrete size choices. Similar to the results in Table 1 we observe that due to the snapping mechanism, there is a perceivable difference in the capacitance values but the delay and power values remain nearly the same. As mentioned previously, the post-snap capacitance values correspond to specific buffer size choices for the first inserted buffer.

Finally, we also show in Table 2 that our method has faster runtimes over all benchmarks with a reduction of about 5-6X for the larger nets. In contrast to the assertion in the Lillis paper, we found that the addition of leakage into the power function drastically increases the number of distinct values of the power number and in turn reduces the number of prunable candidates dramatically. As specified previously, we have assumed that there are nine discrete buffer size choices in our library. For cell libraries that are more fine grained (with the number of discrete buffer choices increasing by 2-3X) our method will maintain runtime efficiency and will provide an even larger runtime improvement over traditional discrete buffer insertion.

6 Conclusion

In this paper we have presented a polynomial time $O(n^2)$ algorithm that performs buffer insertion while constructing optimal candidate surfaces in the delay, capacitance and power (DCP) space. In our formulation, we consider a continuously sized buffer library and first characterize the different buffers according to their specific sizes. We then use this characterization and perform a van Ginneken style buffer insertion to generate a power optimal set of locations and sizes for the different buffers that need to be inserted into the sample net. For each inserted buffer we optionally snap to the nearest sized device in the library. We observe that for a library with sufficient granularity, the error introduced due to the snap correction mechanism is minimal. We present test results for a set of large benchmark circuits and demonstrate the efficacy of the proposed method compared to that of traditional discrete buffer insertion algorithms. We observed that compared to the previous power aware buffer insertion approach, we obtain a runtime speedup of about 5-6X.

7 Acknowledgments

This research work was supported by NSF, DARPA and GSRC.

8 References

[1] R. H. J. M. Otten, R. Brayton, "Planning for performance", *DAC* 1998.

[2] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal elmore delay", *ISCAS* 1990.

[3] C. J. Alpert, A. Devgan, "Wire segmenting for improved buffer insertion", *DAC* 1997.

[4] C. J. Alpert, A. Devgan, S. Quay, "Buffer insertion for noise and delay optimization", *DAC* 1998.

[5] C. Chu, D. F. Wong, "A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing", *IEEE Trans. on CAD*, 18(6), 1999.

[6] C.-P. Chen, C. Chu, D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation", *IEEE Trans. on CAD*, 18(7), 1999.

[7] J.-L. Tsai, T.-H. Chen, C.-P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing", *IEEE Trans. on CAD*, 23(4), 2004.

[8] C. J. Alpert, C. Chu, G. Gandham, M. Hrkic, J. Hu, C. Kashyap, S. Quay, "Simultaneous driver sizing and buffer insertion using a delay penalty estimation technique", *IEEE Trans. on CAD*, 23(1), 2004.

[9] J. Lillis, C.-K. Cheng, T. Y. Lin, "Simultaneous routing and buffer insertion for high performance interconnect", *GLSVLSI* 1996.

[10] H. Zhou, D. F. Wong, I. Liu, A. Aziz,"Simultaneous routing and buffer insertion with restrictions on buffer locations", *IEEE Trans. on CAD*, 19(7), 2000.

[11] S. Dechu, Z. Shen, C. Chu, "An efficient routing tree construction algorithm with buffer insertion, wire sizing and obstacle considerations", *ASP-DAC* 2004.

[12] R. Li, D. Zhou, J. Liu, X. Zeng, "Power-optimal simultaneous buffer insertion/sizing and wire sizing", *ICCAD* 2003.

[13] A. Murugavel, N. Ranganathan, "Gate sizing and buffer insertion using economic models for power optimization", *International Conf. on VLSI Design*, 2004.

[14] S. Weiping, L. Zhuo, "An O(nlogn) time algorithm for optimal buffer insertion", *DAC* 2003.

[15] P. Saxena, N. Menezes, P. Cocchini, D. Kirkpatrick, "Repeater scaling and its impact on CAD", *IEEE Trans. on CAD*, 23(4), 2004.

[16] K. Bernstein, C.-T. Chuang, R. Joshi, R. Puri, "Design and CAD challenges in sub-90nm technologies", *ICCAD* 2003.

[17] P. Saxena, N. Menezes, P. Cocchini, D. Kirkpatrick, "The scaling challenge: Can correct-by-construction design help?", *ISPD* 2003.

[18] S. Narendra, S. Borkar, V. De, D. Antoniadis, A. Chandrakasan, "Scaling of stack effect and its application for leakage reduction", *ISLPED* 2001.

[19] J. Lillis, C.-K. Cheng, T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model", *JSSC*, 31(3), 1996.

[20] J. Shah, S. Sapatnekar, "Wire sizing with buffer placement and sizing for power-delay tradeoffs", *International Conf. on VLSI Design*, 1996.

[21] J. Fishburn, C. Schevon, "Shaping a distributed RC line to minimize Elmore delay", IEEE Trans. on Circuits and Systems, 42(12), 1995.

[22] C. Chu, D. Wong, "Closed form solution to simultaneous buffer insertion/sizing and wire sizing", *TODAES* 2001.

[23] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, Y. Watanabe, "A delay model for logic synthesis of continuously-sized networks", *ICCAD* 1995.

[24] W. Chen, C.-T. Hsieh, M. Pedram, "Simultaneous gate sizing and fanout optimization", *ICCAD* 2000.

[25] W. C. Elmore, "The transient response of a damped linear network with particular regard to wideband amplifiers", *J. Applied Physics*, 19, 1948.

[26] F. Beeftink, P. Kudva, D. Kung, L. Stok, "Gate-size selection for standard cell libraries", *ICCAD* 1998.

[27] R. Haddad, L. P. P. P. van Ginneken, N. Shenoy, "Discrete drive selection for continuous sizing", *ICCD* 1997.

[28] D. S. Kung, "A fast fanout optimization for near-continuous buffer libraries", *DAC* 1998.

[29] W. Chen, C.-T. Hsieh, M. Pedram, "Simultaneous gate sizing and fanout optimization", *ICCAD* 2000.

[30] GNU Scientific Library (GSL), http://www.gnu.org/software/gsl/.

[31] C. J. Alpert, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, A. J. Sullivan, P. Villarrubia, "Buffered Steiner trees for difficult instances", *IEEE Trans. on CAD*, 21(1), 2002.

[32] http://dropzone.tamu.edu/~cnsze/GSRC/ctree.html

[33] International Technology Roadmap for Semiconductors (ITRS), 2001.