A Placement Methodology for Global Interconnect Reduction and Its Impact on Performance

Andrew B. Kahng and Igor L. Markov and Sherief Reda

Abstract

Global interconnects are a bottleneck in today's high-performance deep sub-micron designs. In this paper, we propose a modification to the top-down min-cut placement algorithm to reduce the number of global interconnects. Our method is generic and does not involve any timing analysis during or prior to placement. In essence, we skew the netlength distribution produced by a min-cut placer so as to reduce the number of long nets, with minimal impact on the overall wirelength. Empirically this approach has a negligible impact on placement runtime, but leads to a significant reduction in the number of global interconnects. The fewer interconnects translate to about 25% savings in the number of buffers required for signal integrity and electrical sanity, and also improve timing as measured by the worst negative slack and total negative slack of industrial benchmarks by up to 70% compared to traditional min-cut placement flow (e.g., Capo 8.7 [9])[1].

I. Introduction

Global interconnects are harmful [22], [30] since they (i) severely increase RC propagation delay which is quadratic in wirelength, (ii) likely contribute to critical paths that form performance bottlenecks, (iii) require buffering for signal integrity and electrical sanity, e.g., slew rate control, even if they do not participate in critical paths, with state-of-the-art designs routinely requiring thousands of buffers for signal integrity [4], (iv) increase power dissipation, and (iv) can have time-of-flight that is equivalent to several clock cycles which necessitates flipflop insertion. These detrimental effects get more pronounced as scaling in deep sub-micron continues.

Techniques proposed in the past attempt to alleviate those effects of global interconnects by providing solutions that deal with existing interconnect structures, e.g., by using fat wires and buffering [27], [22], [30]. Buffering, for example, will almost linearize the delay of a long interconnect, but at the expense of an additional number of buffers. These buffers represent an additional source of power consumption to the design. In contrast, our work seeks to decrease the number of long interconnects. We propose a placement methodology that directly leads to fewer global interconnects. Min-cut placement is widely used by today's state-of-the-art placement tools since (i) its performance is scalable, and (ii) it leads to high quality placements as measured by the wirelength objective. The main objective of a min-cut algorithm is to minimize the total wirelength via sequential minimization of cut values. While minimizing

A. B. Kahng is with the Departments of Computer Science and Engineering, and of Electrical and Computer Engineering, UC San Diego, La Jolla, CA 92093-0114.
Igor L. Markov is with the EECS Department, University of Michigan Ann Arbor, MI, 48109.
S. Reda is with the Department of Computer Science and Engineering, UC San Diego, La Jolla, CA 92093-0114.
E-mail: abk@ucsd.edu, imarkov@eecs.umich.edu, sreda@cs.ucsd.edu.

[1]Earlier results of this work have been reported at Design, Automation and Test in Europe 2004. New results in this manuscript include detailed experiments on wirelength distribution and buffer insertion.

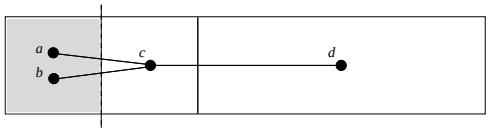


Fig. 1. Min-cut placements can unnecessarily lead to large number of global interconnects.

the total wirelength is helpful in meeting the routing supply and reducing the total power consumption, it may lead to placements with large number of global interconnects. This is illustrated in Figure 1, where a min-cut partitioner would favor moving node *c* to the left shaded block to reduce the cut value, but leading to a long interconnect between nodes *c* and *d*.

On the other hand, many analytical placers [11], [35] minimize the quadratic length objective which has been historically viewed as a compromise between mathematical and computational convenience and the end goal of total wirelength minimization. One advantage of analytical placers is that they may lead to fewer long wires as observed by [28]. The debate regarding the use of linear versus quadratic wirelength goes back to at least the mid-1980s [32], [28], [14], [18].

In this paper, we propose a new methodology to reduce the number of global interconnects produced by top-down min-cut placers and examine the impact of such reduction on the performance as measured by the number of buffers required for signal integrity and the design timing. In our approach, we modify net weights during min-cut partitioning by giving previously cut nets a higher priority. Such nets are less likely to be cut in the future, and this tends to reduce their final wirelength. We call our adjustments to net weights *boosting factors* and compute them dynamically based on lower bounds on net length during min-cut placement (using upper bounds is equivalent, as we show below). This results in significantly fewer global interconnects with little impact on total wirelength. Empirical validation involves calculating the number of buffers [4] required for signal integrity, e.g., slew rate control, in boosted and non-boosted designs. We also route the placements using a commercial router and execute static timing analysis using Cadence's Pearl (version 5.1) to measure the impact of boosting on timing. The conclusion is that our technique leads to fewer global interconnects, improving both the number of buffers and the timing.

The organization of this paper is as follows. Section 2 provides a brief overview of related work in the literature. Section 3 gives basic definitions and motivates our work. Section 4 introduces the concept of boosting, determines which nets should be boosted and discusses the effect of boosting on cut size. Empirical validation is covered in Section 5, and Section 6 summarizes our contributions.

II. Previous Work

Several related techniques give important context to this work. Most timing-driven placement methods start by identifying nets that belong to critical paths. Various methods are then used to control the net lengths [20], [29], [31], [26], [11], [24], [15], [16]. Main differences are associated with whether the placement methodology is analytical [29], [31], [11] or top-down [20], [24], [15], [16]. Top-down methodologies use quadratic partitioners [24] or min-cut partitioners [20], [16]. Net length constraints are typically translated into net weights [20], [24] or handled as upper bounds [29], [15].

Ou and Pedram [24] control the number of cuts in a critical path, since the more cuts a path experiences, the longer the path tends to be. Hence, the method of [24] gives large weights to critical nets and imposes an upper bound on the maximum number of times a path can be cut. In [15], linear programming is used to constrain the bounding boxes of critical nets.

Marek-Sadowska and Lin [20] perform static timing analysis at each level of recursive bisection and then translate slacks to net weights. These weights affect the min-cut partitioner and eventually reduce signal delay. Another recent approach that uses a min-cut partitioner is due to Kahng *et al.* [16]. Consider a block under partition. Some of the block's cells are pre-assigned and fixed to the child partitions so as to reduce the negative slack of critical paths. After cell pre-assignment, the hypergraph partitioner is invoked on the remaining cells.

Buffering algorithms can be classified according to the objective of buffering. Alpert *et al.* [4] addresses the problem of finding a minimum-cost buffered routing tree such that the capacitive load of each buffer does not exceed the buffer specification. This is critical for meeting slew rates constraints, a major bottleneck in 130nm designs and below. Buffering for critical nets, e.g., clock net, has the additional constraint that skew constraints must be met [25]. Another important buffering objective is minimizing the signal delay, where critical nets are buffered in order to meet the timing constraints [33], [21], [2].

III. Definitions and Motivating Example

A circuit netlist is represented by a hypergraph $H(V,E)$, where V is the set of nodes corresponding to the circuit cells such that each node $v \in V$ has weight $w(v)$ reflecting its physical area. We refer to the horizontal location of v by v_x and the vertical location by v_y. Hyperedges $E \subseteq 2^V$ model circuit nets, where each hyperedge $e_i \in E$ is a set of nodes that are connected by a net. Associated with each placed hyperedge e_i is a bounding box $BB(e_i)$ whose corners are the four points (e_i^l, e_i^u), (e_i^l, e_i^b), (e_i^r, e_i^u) and (e_i^r, e_i^b), where the superscripts u, b, l and r refer to upper, bottom, left and right respectively. The length of a given net/hyperedge is the half-perimeter of the bounding box, sometimes denoted with HPWL (half-perimeter wirelength).

We think of a placement region as a collection of *blocks* as shown in Figure 2. Each block corresponds to a fixed rectangle into which some (sub)hypergraph vertices should be placed. Initially, the chip area is comprised of one block. The min-cut placement methodology proceeds by recursively partitioning each block and its associated hypergraph, and assigning the partitioned subhypergraphs to sub-blocks. Cut direction usually alternates between vertical and horizontal, or is determined the block aspect ratio [9]. Hence, the product of the partitioning process is a slicing floorplan as illustrated in Figure 2. In this figure, we illustrate two horizontal and vertical cut levels numbered from 1 to 2. The partitioning process continues until a certain block threshold size, beyond which end-case placers are used to assign actual locations for the hypergraph nodes in their corresponding blocks [7], [9], [36]. For each block B_j, we let $BB(B_j)$ represent the bounding box of block B_j, where the bounding box of a block $BB(B_j)$ is the rectangle whose corners are the corners of the block.

As we pointed out earlier, it is desirable to limit the increase of lower bounds of long nets since this limitation reduces their final wirelength, leading to an overall reduction in number of inserted buffers as well as the total delay. The following example further illustrates our point.

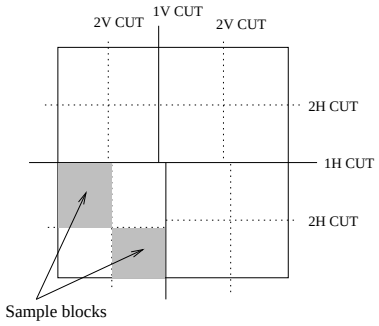


Fig. 2. A slicing floorplan produced by a top-down min-cut placer. Two rounds of vertical and horizontal cuts are illustrated. 1H CUT indicates the first horizontal cut level, while 2H CUT indicates the second horizontal cut level. 1V CUT and 2V CUTS are for vertical cuts.

Example 1: Assume as in Figure 3 that we have three hyperedges e_1, e_2 and e_3 and that we are currently partitioning the shaded blocks. We have two solutions: solution A and solution B. In both solutions, the total wirelength and cuts are both equal to 8. However, we prefer solution B since its total delay is smaller. Solution A has 38 units of delay (in terms of squared length), while solution B has 24 units of delay. Also, if the critical length [22], i.e., the maximum wirelength that a driver can support, is 5, then a buffer is needed for solution A, while solution B needs no buffers.

The length of each net can be upper-bounded and lower-bounded at any point during top-down placement, based on which blocks contain incident cells ("incident blocks"). For convenience, we treat fixed cells, pins and pads as individual blocks. The half-perimeter (HPWL) of a net cannot exceed the common half-perimeter of all incident blocks. Related lower bounds can be defined separately in the horizontal and vertical dimensions as we now describe for the horizontal case. Indeed, the left-most cell on the net must be placed to the left from the right edge of every incident block; similarly, the right-most cell on the net must be placed to the right from the left edge of every incident block (note that we do not need to know which particular cells will eventually be left-most and right-most when the final placement is produced). To lower-bound the horizontal component of the net's half-perimeter, we need to consider right edges of all incident blocks and find the left-most. Similarly, we find the right-most left edge over all incident blocks, and compute the distance between the two horizontal locations. The net's span cannot be smaller than that. A lower bound for net's length is produced by adding the horizontal and vertical components.

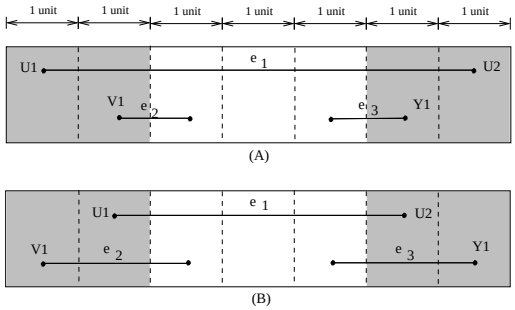
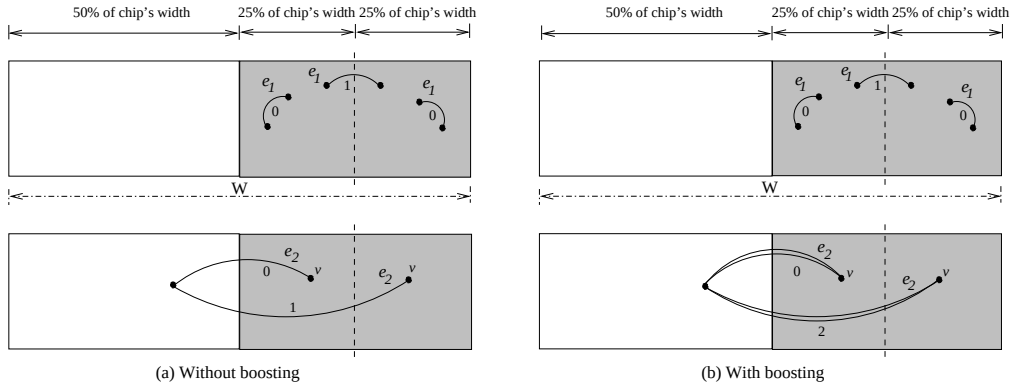


Fig. 3. Two placements with equal linear wirelength but different squared wirelength, which is greater in the (A) example.



(a) Without boosting (b) With boosting

Fig. 4. An illustration of boosting. The hyperedge e_2 was cut at the first vertical cut. The dashed vertical line represents the new second vertical cut. With respect to the new cut, each hyperedge can be cut or uncut as illustrated by the various positions of each hyperedge; we label each possibility by its contribution (0/1) to the cut value as well. We notice that without boosting the partitioner does not differentiate between the two hyperedges. However, if e_2 is cut then its length is lower-bounded by quarter of the chip width. To encourage the partitioner to cut e_1 over e_2, we multiply the weight of e_2 by a factor of 2 as shown in picture (b).

Let us study the progression of lower and upper bounds for nets that are not incident to fixed cells, pins or pads. At the beginning of top-down placement the upper bound for every such net is the half-perimeter of the core area, and the lower bound is zero. At every cut, the upper bound decreases for every uncut net and does not change for every cut net. Similarly, the lower bound increases for every cut net, and does not change for every uncut net. Thus, in general, lower bounds gradually increase and upper bounds gradually decrease until they meet at the end of top-down placement, at which point their values are both equal to the net's actual length.

IV. Achieving Fewer Global Interconnects with Min-Cut Placers

In this section we outline a potential methodology to reduce the number of global interconnects. In the process we highlight one of min-cut placement drawbacks that leads to the formation of such interconnects.

A. Boosting Min-Cut Placement

Most netlists allow multiple placements with equal total wirelength, however in performance-driven placement it is important to prevent very long nets as we discussed earlier. Traditional min-cut placers may end up producing several long nets so as to shorten medium-sized nets, and often leave such trade-offs up to chance. This suggests the possibility of useful tie-breaking that would not affect runtime or solution quality, but may address additional design objectives. Motivated by this, we propose a way to discourage very long nets by dynamically increasing the weight of each net after it has been cut, so as to prevent further cuts.

Below we assume a placer that partitions each block in approximately equal sub-blocks (this is typically true for sufficiently large blocks in the absence of significant design constraints).

Example 2: Let us examine the first two vertical cuts (and skip horizontal cuts if any), which may be at 50% and 75% of the chip width, as shown in Figure 4. If a hyperedge is cut twice, its length can be as high as the chip width. However, if it is cut only once, its length can be upper bounded by either 50% of the chip width (cut by the second

cut) as hyperedge e_1 or by 75% of the chip's width (cut by the first cut) as hyperedge e_2. In this case, our proposed modification to the standard min-cut framework applies after the first cut. Namely, the weights of cut hyperedges are increased so as to discourage them from being cut again. This way we may achieve a 75% upper bound for the net length of many such hyperedges, while the length of each previously uncut hyperedge is already subject to a 50% upper bound.

In our example, we can also track lower bounds for net lengths. Indeed, two cuts imply a 25% lower bound, while any one cut implies only a 0% lower bound. Given how *upper* bounds change, it appears that increasing the weights of cut hyperedges may prevent very long nets. Furthermore, given how *lower* bounds change, it appears that if we do not prevent multiple cuts early in top-down placement, it may be difficult to prevent long nets.

In practical terms, we multiply the weight of the hyperedge e_2 by the factor $\beta = 2$ as shown in Figure 4 (b). This strengthens the connection between the nodes of hyperedge e_2 encouraging the partitioner to move node v to the left shaded sub-block and consequently improving the upper bound on the hyperedge's length by a quarter of the chip width. To formalize the idea of increasing the weight of longer nets, we define *boosting* as follows.

Definition 1: A hyperedge $e_i \in E$ is *boosted* by multiplying e_i's weight by a certain factor, i.e., the *boosting factor* β, and a hyperedge is boosted only if partitioning the current block can increase the lower bound on the hyperedge span (HPWL) if the hyperedge is cut, or equivalently only if partitioning the current block can reduce the upper bound on the net hyperedge span if it is not cut.

We now propose *eligibility* conditions for boosting a hyperedge. Given a block B_j and a corresponding hypergraph $H(V,E)$ to be partitioned, a hyperedge $e_i \in E$ is *eligible* for boosting only if it meets all of the following conditions:

1. $BB(e_i) \not\subseteq BB(B_j)$, i.e., e_i has been cut before.
2. There exists some node v of e_i in $BB(B_j)$ such that

   If B_j is cut vertically:

   - either $v_x = e_i^l$ and no point of the line segment $[(e_i^r, e_i^b), (e_i^r, e_i^u)]$ is in $BB(B_j)$.
   - or $v_x = e_i^r$ and no point of the line segment $[(e_i^l, e_i^b), (e_i^l, e_i^u)]$ is in $BB(B_j)$.

   If B_j is cut horizontally:

   - either $v_y = e_i^u$ and no point of the line segment $[(e_i^r, e_i^b), (e_i^l, e_i^b)]$ is in $BB(B_j)$.
   - or $v_y = e_i^b$ and and no point of the line segment $[(e_i^r, e_i^u), (e_i^l, e_i^u)]$ is in $BB(B_j)$.

We illustrate these conditions by the following example.

Example 3: Figure 5 illustrates the eligibility for boosting in four sample cases:

- Case (i): A hyperedge of degree three. The hyperedge is eligible for boosting since partitioning the current (shaded)

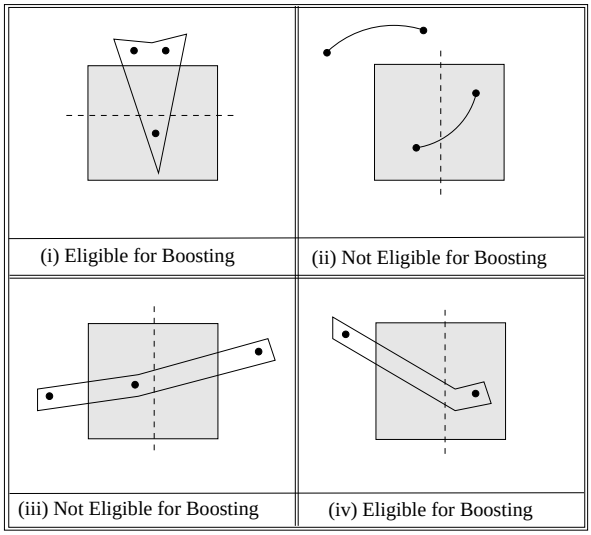


Fig. 5. Illustrative cases for the eligibility of hyperedge boosting. The shaded block represents a block under partitioning by the vertical dashed line. A hyperedge is eligible for boosting if partitioning the block can increase the lower bound on its length. Case (i) is a hyperedge of degree three, with one node inside the block under partition, while the two other nodes are outside. Case (ii) is two hyperedges each of degree two. One hyperedge has both of its nodes inside the block under partition, while the other hyperedge has both of its nodes outside the block. Case (iii) is a hyperedge of degree three, with one node inside the block, while the other two nodes are outside. Case (iv) is a hyperedge of degree two, with one node inside the block under partition and one outside.

block can extend the lower bound on the hyperedge length.

- Case (ii): Two hyperedge each of degree two. The two hyperedges are not eligible for boosting since partitioning the current block will not affect on the lower bound of their length. Condition 2 is violated.
- Case (iii): A hyperedge of degree three. The hyperedge is not eligible for boosting since the position of the node contained within the current block does not affect the hyperedge span. Condition 2 is violated. For partitioning purposes this hyperedge can be ignored, since partitioning does not affect its half-perimeter wirelength. Such hyperedges are called “inessential” [7].
- Case (iv): A hyperedge of degree two. The hyperedge is eligible since partitioning the current block can increase the lower bound.

While we give the general conditions - unrelated of a particular min-cut placer - necessary for boosting a hyperedge, we note that handling of external hyperedges is strongly related to the terminal propagation process [10]. For example, in Capo [9] hyperedges of Case (iii) of the previous example are automatically handled by the underlying terminal propagation mechanism [8], where such hyperedges are omitted during the partitioning of the shaded block.

While Definition 1 gives eligibility conditions for boosting a hyperedge, this does not mean that we should necessarily boost the hyperedge. We introduce a further condition that helps to boost only edges that may become global. If we assume an alternating horizontal-vertical cut sequence and we examine the vertical cut sequences[2] then the first vertical cut bisects the chip width. If the chip width is W then the first vertical cut is at $\frac{W}{2}$, creating 2 blocks. The

[2]Horizontal cuts are treated similarly.

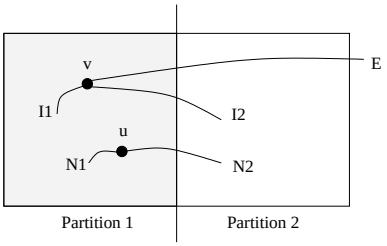


Fig. 6. The effects of boosting on cut size. Node v is connected to the sets of hyperedges I_1, I_2, E_1, and E_2 (these are not individual hyperedges). Node u is connected to the two sets of hyperedge N_1 and N_2.

second-level vertical cuts are at $\frac{W}{4}$ and $\frac{3W}{4}$, creating four blocks, and the i^{th} level cuts create 2^i blocks each of width $\frac{W}{2^i}$. This process continues until all blocks reach a certain threshold.

If a hyperedge is cut at a certain cut level then nodes of this hyperedge exist on *both* sides of the cut. Hence, if a hyperedge is first cut at cut level l and and later at cut level $l+1$ then this increases the lower bound on its horizontal length by $W/2^{l+1}$. Suppose that a hyperedge was cut at levels l through $l+\tau$, where $\tau \geq 1$, then the lower bound on the hyperedge's length is $\sum_{i=l+1}^{l+\tau} W/2^i = \frac{W\cdot(2^\tau-1)}{2^{l+\tau}}$, where the contribution of level $l+\tau$ to the lower bound is $\frac{W}{2^{l+\tau}}$. This means that the contribution of new cuts is exponentially decreasing as we descent in the top-down hierarchy. In the circuits we experiment with, we have found that there is little point in boosting after $\tau \geq 4$ since the total contributions of new cut levels (if the hyperedge ever gets cut again) will not exceed 7% of the hyperedge's horizontal length at $\tau = 4$. We note however that in reality, cut-sequences may deviate from alternating due to (i) layout aspect ratios that are different from the unity, (ii) min-cut partition sizes might not be balanced. In our experiments, we have found that it is of no benefit to boost a hyperedge for more than eight levels (four horizontal placement levels and four vertical placement levels), since block sizes after eight placement levels are typically less than few percentage of the overall chip's core size, and boosting after this will effectively hurt the cutsize without any prospect of reducing the number of global interconnects. We also note that after each placement level, we reset or *deboost* any boosted hyperedges so that the boosting weights do not increasingly accumulate in any hyperedge.

Another possibility is to only boost hyperedges that are timing critical in a manner similar to Ou and Pedram's [24] scheme. However, limiting boosting to such cases will likely lead to miss some non-timing critical global interconnects and consequently a larger number of buffers for the sake of signal integrity [4].

B. Effect of Boosting on Cut Values

In the previous subsection, we determined the eligibility conditions for boosting a hyperedge. However, a crucial parameter is the value of the boosting factor β. A high value for β is expected to encourage a min-cut partitioner to move nodes so as to reduce the hyperedge's span. In the following, we analyze the relation between β and the cut size, which directly affects the final HPWL [6].

Suppose that we have two nodes v and u as shown in Figure 6, where the rectangle represents the current block under partition. We say that a hyperedge is *external* if some of its nodes lie outside the current block under partition, and a hyperedge is *internal* if all of its nodes lie within the block under partition. Node v is connected to a set E of external hyperedges eligible for boosting, and to the sets of internal hyperedges sets I_1 and I_2. Node u is connected

to only the internal hyperedges sets N_1 and N_2. Thus, the gain of moving node v from partition 1 to partition 2 is $\delta_v = |I_2| - |I_1| + |E|$, while the gain of moving node u is $\delta_u = |N_2| - |N_1|$. With boosting, the gain of node v becomes $\delta_v^\beta = |I_2| - |I_1| + \beta|E|$, while the gain of node u remains the same since it is not connected to any hyperedges eligible for boosting. We now distinguish three important cases for a min-cut partitioner operating with boosting:

- Case 1: $\delta_u = \delta_v$ and $\delta_v^\beta > \delta_u$. In this case the partitioner ends up moving node v rather than u. Hence, boosting does not worsen the cut size but at the same time the span of the long external hyperedges is reduced.
- Case 2: $\delta_u > \delta_v$ and $\delta_v^\beta < \delta_u$. Here the partitioner ends up moving node u rather than v as if it is not operating under boosting. Hence, boosting does not worsen the cut size but there is no benefit from boosting either.
- Case 3: $\delta_u > \delta_v$ but $\delta_u > \delta_v^\beta$. In this case the partitioner moves node v rather than u, effectively reducing the spans of the external hyperedges but worsening the cut size by $|N_2| - |N_1| - |I_2| + |I_1| - |E|$ in comparison to moving node u.

Under any scheme that boosts net weights only for small values of τ (the maximum amount of levels a hyperedge can be boosted), the likelihood of Case 3 is small. To see this, we notice that $\delta_u - \delta_v = |N_2| - |N_1| - |I_2| + |I_1| - |E| > 0$, however with boosting, $\delta_u - \delta_v^\beta = |N_2| - |N_1| - |I_2| + |I_1| - \beta|E| < 0$. Thus the partitioner thinks that moving node v is more beneficial. We notice that the difference between $\delta_u - \delta_v$ and $\delta_u - \delta_v^\beta$ is $(\beta - 1)|E|$. Thus, the likelihood of case 3 increases as the value $(\beta - 1)|E|$ increases in magnitude. If $\beta = 2$, this implies that the likelihood of case 3 increases as the number of boosted edges a node is connected to increases. However, during the first few levels - where boosting is applicable - in min-cut placement, there are typically few edges that are boosted, and the likelihood that a node is is connected to more than one or two boosted hyperedges is typically low. For example, in a typical benchmark, the IBM01, the number of nodes connected to boosted edges for some placement levels are as follows:

- Placement level 1: 2.59% of the nodes are connected to a single boosted edge. 0.56% of the nodes are connected to two boosted edges. 0.08% of the nodes are connected to 3 or more boosted edges.
- Placement level 2: 10.81% of the nodes are connected to a single boosted edge. 2.45% of the nodes are connected to two boosted edges. 0.96% of the nodes are connected to 3 or more boosted edges.
- Placement level 3: 16.80% of the nodes are connected to a single boosted edge. 5.41% of the nodes are connected to two boosted edges. 2.67% of the nodes are connected to 3 or more boosted edges.

Thus, the common case is that each node is connected to very few boosted hyperedges, leading to a graceful reduction in the cut size. We tabulate the total cut value per placement level for a number of benchmarks in Table V.

V. Experimental Validation

In this section we empirically assess the effect of boosting on the interconnect wirelength distribution, and consequently the number of buffers and the circuit timing as measured by the worst negative slack and the total negative slack (TNS). Our implementation is based on Capo [9] (version 8.7) and is compared to (i) the unmodified Capo 8.7, (ii) Cadence's QPlace place (version 5.2). We use the buffering algorithm of Alpert *et al.* [4] to calculate the number of buffers that would be inserted fr multi-pin nets. Cadence's WarpRoute (version 5.4) is used for global and detailed routing, and Cadence's Pearl (version 5.1) is used for static timing analysis.

The four industrial benchmarks used for our experiments are described in Table I. The number of cells in these

benchmark	cells	nets	core region	whitespace	metal layers	clock period (ns)
A	40347	42487	2622.00 x 2456.00	42.5%	4	16.60
B	21103	21230	2142.20 x 1969.40	13.5%	4	4.545
C	33917	39153	2315.70 x 2315.70	49.8%	4	29.6
D	9585	10398	8705.40 x 8696.80	29.7%	5	36.67

TABLE I

BENCHMARK CHARACTERISTICS.

designs are in the range of $9-40k$. While these benchmarks do not represent the state-of-the-art, these are the only industrial benchmarks with timing information that are accessible to the authors. Other publicly available benchmarks, e.g., IBM benchmarks, lack timing information necessary for some of our experiments.

The experimental results section is organized as follows. We first examine the effect of boosting on the wirelength distribution; we show that boosting leads to fewer global interconnects. Then we calculate the number of buffers that would be inserted in non-boosted and boosted placements. Our calculations indicate that boosting significantly reduces the number of buffers. Finally, we examine the impact of boosting on the timing characteristics of the circuit.

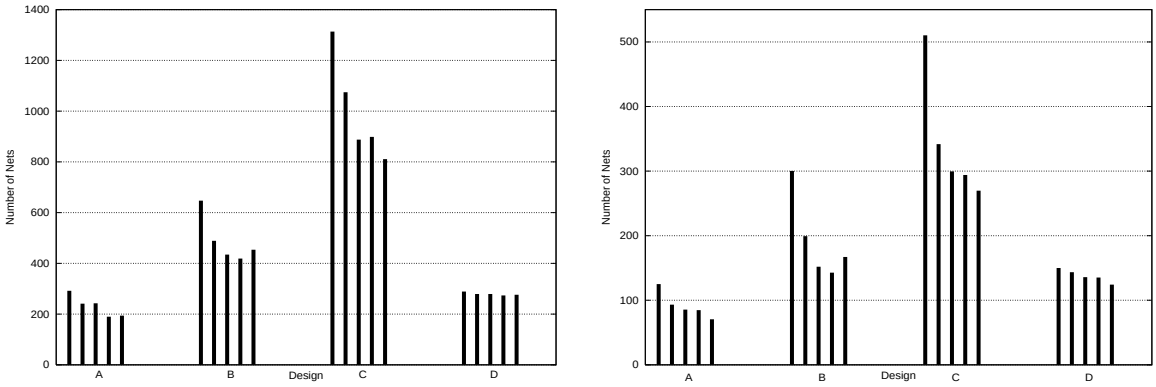
A. Effect of boosting on the wirelength distribution

In a first set of experiments, we verify our premise that boosting alters the interconnect distribution leading to fewer global interconnects. We compare the wirelength distribution produced from regular Capo and boosted Capo for different boosting factors. To report the wirelength distribution:

1. We calculate the half perimeter of the chip's surface available for standard cells and blocks.
2. A net histogram is constructed in units of tenth of the chip's half perimeter, i.e., we have a histogram of ten bins, where a net belongs to the i^{th} bin if its HPWL is $\frac{i}{10}$ that of the chip's half perimeter.
3. The number of nets within each bin of the histogram is reported.

The wirelength distribution results are given in Table II for five different boosting factors: 1, 2, 3, 4, and 5. The use of integer boosting factors is necessitated by the fact that multi-level partitioners [3], [17] - the core engines of min-cut placers - use FM partitioning [13]. FM partitioning relies on an underlying integral discretized bucketing scheme. If fractional boosting values were to be used, it would require rounding, which produces inaccurate results. The results in Table II are averaged for four random seeds. Notice that a boosting factor of 1 corresponds to the regular min-cut placement. The table shows a clear trend in all benchmarks: boosting alters the wirelength distribution leading to fewer longer nets, furthermore, the larger the boosting factor, the fewer global nets that are produced. We also report the total HPWL to estimate the increase in wirelength. In general, boosting leads to a slight increase in wirelength.

The results of Table II are summarized in Figures 7.a and 7.b where the number of nets that span more than 20% and 30% of the chip's half perimeter are plotted for the different boosting factors. In the bar plot of each benchmark, the boosting factor increases as we move from the left to the right. We notice that a boost factor of 2 produces the sharpest reduction in the number of global interconnects. A boosting factor of 2 reduces the number of global interconnects as



(a) Number of interconnects that span more 20% of the chip's half perimeter.

(b) Number of interconnects that span more 30% of the chip's half perimeter.

Fig. 7. The number of global interconnects as a function of the boosting factor for various designs. For each benchmark, we plot the number of interconnects that span more than 20% (subfigure a), and 30% (subfigure b) of the chip's half perimeter. We plot these values for five different boosting factors ($\beta = 1, 2, 3, 4, 5$), where the value of β increases as we move from left to right in the bar plots of each benchmark.

measured by the 20% rule by 17%, 24%, 18%, and 3% for the four benchmarks, and reduces the number of global interconnects as measured by the 30% rule by 25%, 33%, 33%, and 5%.

B. Effect of boosting on number of buffers

In a second set of experiments, we calculate the number of the buffers that would be inserted in a boosted placement versus a non-boosted placement. A critical quantity in estimating the number of buffers is $l_{critical}$ which represents the maximum amount of wirelength (or capacitance) that a buffer can drive [22]. If P represents a net's HPWL then $\lfloor \frac{P}{l_{critical}} \rfloor$ represents a lower bound on the number of required buffers. This lower bound is exact for 2-pin nets. For multi-pin net, we use the buffering algorithm by Alpert *et al.* [4] to calculate the exact number of buffers. Given the pin locations of a net, the algorithm constructs a buffered Steiner tree using a clustering heuristic [4] which is shown to produce near-optimal number of buffers. To avoid tuning our results to a particular technology, we calculate the number of buffers for wide-range values of critical lengths, i.e., we start with $l_{critical}$ equal to 10% of the chip's half perimeter and gradually increase $l_{critical}$ in steps of 5% until 100% of the chip's half perimeter. Our results are reported in Table III.

From the table, it is evident that boosting consistently reduces the number of buffers for all values of critical lengths, with average improvements of 21.7%, 21.75%, 22.77%, and 14.5% for the 4 designs respectively. The percentage improvement slightly increases as the critical length increases. These reductions in number of buffers lead to overall savings in die area, power consumption, and congestion. The results in Table III are summarized in Figure 8.

We also repeat the same experiments for the IBM (version 2) benchmarks [36] and report the results in Table IV. We also report the number of buffers required for two other academic placers Dragon [36] (version 3.1) and FengShui [19] (version 2.6). It is evident that boosting consistently reduces the number of buffers at a small cost to total HPWL.

C. Effect of boosting on timing

In a third set of experiments, we examine the effect of boosting on circuit timing. These experiments are performed with industrial tools in a realistic design flow. Table VI reports our results which are average of four random runs. The alternate design flows in this table are as follows. In the flow **indust NTD**, the industrial placer is used to place the benchmark in a non-timing driven mode and then WarpRoute (version 5.4) is used to route the benchmark in a timing-driven mode. In the flow **indust TD**, the industrial placer is used to place the benchmark in a timing-driven mode and then WarpRoute is used to route the benchmark in a timing-driven mode. In the flow **CAPO regular**, Capo 8.7 is used to place the benchmark and then WarpRoute performs timing-driven routing on Capo's output placement. In the flow **CAPO BOOST**, we use a modified version of Capo and route the resulting placements using WarpRoute in timing-driven mode. In Table VI, we report the the Half-Perimeter Wirelength (HPWL), **VIOLATIONS** is the number of detailed routing violations, i.e., number of nets that are not completely routed successfully. **time** is the placement runtime in seconds, **Wirelength** is the actual wirelength as reported by WarpRoute, **SLACK** is the worst negative slack as reported by the analysis using Pearl (version 5.2), **TNS** is the Total Negative Slack of all nets that have negative slack.

Taking into account that Boosting is a timing oblivious technique (so as Capo), we find that boosting improves Capo's circuit timing as indicated by the data in Table VI. For example, the negative slack of Design B improves by about 43% in comparison to Capo, and furthermore boosting reduces the TNS by about 78% in comparison to Capo. Also, boosting improves the worst negative slack of Design C by 60% in comparison to Capo, and furthermore the TNS is improved by about 61%. For designs A and D, none of the placers is able to exploit any advantage in their timing-driven mode. As indicate in the table, there are a small number of routing violations with some test cases. These can be handled by logic transformations or manually. We also notice that boosting increases the placement runtime by a negligible amount. This is understandable given that before each placement level, nets that meet the boosting eligibility conditions must have their weights increased.

VI. Conclusions

Our work improves min-cut placers by reducing their outcome of global interconnects. These interconnects degrade performance in several ways and are a bottle in 130nm and below designs. The novel technique we propose (boosting) is based on dynamically changing net weights during top-down placement. We first observe that the length of each net is subject to a series of increasing lower bounds that depend on when the net is cut. Therefore we increase the weights of nets with the highest lower bounds, so as to discourage cutting them in the future. This limits the further increase of their lower bounds.

While the changes in weights do not significantly affect the resulting half-perimeter wirelength, they tend to reduce the number of global interconnects, therefore reduce the number of buffers required for signal integrity and generically improve circuit delay as well. To validate this empirically, we implement boosting within the well-established min-cut placer Capo, calculate the new wirelength distributions, calculate the number of the buffers that would be inserted, perform detailed routing, and evaluate timing with Cadence's Pearl (version 5.1). Our experimental results clearly

indicate a consistent and significant reduction in the the number of global interconnects. As a result, we improve circuit timing in several industrial benchmarks compared to both Capo and a leading industrial placer, and also reduce the number of buffers that would be inserted. It is surprising that such an improvement in timing and buffering can be achieved, across several real-world circuits, without giving timing constraints to the placer and with only a slight runtime penalty.

REFERENCES

[1] S. N. Adya et al., "Benchmarking for Large-Scale Placement and Beyond", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 2003, pp. 95-103.

[2] C. Alpert, A. Devgan and S.T. Quay,"Buffer Insertion for Noise and Delay Optimization," *IEEE Transactions on Computer-Aided Design*, vol. 18(11), 1999, pp. 1633–1645.

[3] C. J. Alpert, J. H. Huang, and A. B. Kahng, "Multilevel Circuit Partitioning," in *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 530–533.

[4] C. Alpert, A. B. Kahng, B. Liu, I. Mandoiu and A. Zelikovsky, "Minimum-Buffered Routing of Non-Critical Nets for Slew Rate and Reliability Control", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 2001, pp. 408-415.

[5] C. Alpert, G. Nam and P. Villarubia, "Free Space Management for Cut-Based Placement", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 2002, pp. 746-751.

[6] M. A. Breuer, "Min-Cut Placement", *Journal of Design Automation and Fault Tolerant Computing*, vol. 1(4), 1977, pp. 343-962.

[7] A. Caldwell, A. B. Kahng and I. Markov, "End-Case Placers for Standard-Cell Layout", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 1999, pp. 90-96.

[8] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout", *IEEE Transactions on Computer-Aided Design*, vol. 19(11), 2000, pp. 1304–1313.

[9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 477-482.

[10] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits", *IEEE Transactions on Computer-Aided Design*, vol. 4(1), 1985, pp. 92–98.

[11] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 269-274.

[12] W. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers", *Journal of Applied Physics* vol.19, 1948, pp. 55-63.

[13] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175–181.

[14] L. Hagen and A. B. Kahng, "Improving the Quadratic Objective Function in Module Placement", *Proc. IEEE Intl. ASIC Conf.*, 1992, pp. 171-174.

[15] B. Halpin, C. Y. Roger Chen and N. Sehgal, "Timing Driven Placement Using Physical Net Constraints", *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 780-783.

[16] A. B. Kahng, S. Mantik and I. L. Markov, "Min-Max Placement for Large-Scale Timing Optimization", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 2002, pp. 143-148.

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

[18] J.-M. Li et al., "New Spectral Linear Placement and Clustering Approach", *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 88-93.

[19] M. Yildiz and P. Madden, "Global Objectives for Standard-Cell Placement," in *Proc. IEEE Great Lakes Symposium on VLSI*, 2001, pp. 68–72.

[20] M. Marek-Sadowska and S.-P. Lin, "Timing Driven Placement", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1989, pp. 94-97.

[21] T. Okamoto and J. Cong, "Buffered Steiner Tree Construction With Wire Sizing for Interconnect Layout Optimization," *IEEE Interational Conference on Computer-Aided Design*, 1996, pp. 44–49.

[22] R. Otten, "Global wires: Harmful?,", *Proc. Intl. Symp. Physical Design*, 1998, pp. 104-109.

[23] R. Otten and R. Brayton, "Planning For Performance", *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 122-127.

[24] S. L. Ou and M. Pedram, "Timing-Driven Placement Based on Partitioning with Dynamic Cut-Net Control", *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 472-476.

[25] S. Pullela, N. Menezes, J. Omar and L.T. Pillage, ”Skew and Delay Optimization for Reliable Buffered Clock Trees, ” *IEEE Interational Conference on Computer-Aided Design*, 1993, pp. 556–559,

[26] B. Riess and G. Ettelt, “SPEED: Fast and Efficient Timing Driven Placement”, *Proc. IEEE Intl. Symp. Circuits and Systems*, 1995, pp. 377-380.

[27] G. Sai-Halasz, “Performance Trends in High-End Processors,” *Proc. IEEE*, vol. 83(1), 1995, pp. 20–36.

[28] G. Sigl, K. Doll and F. M. Johannes, “Analytical Placement: A Linear or a Quadratic Objective Function?”, *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 427-431.

[29] A. Srinivasan, K. Chaudhary and E. S. Kuh, “RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs”, *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1991, pp. 48-51.

[30] D. Sylvester and K. Keutzer, “A Global Wiring Paradigm for Deep Submicron Design,” *IEEE Trans. on Computer-Aided Design*, vol. 19(2), 2000, pp. 242–252.

[31] R. Tsay and J. Koehl, “An Analytical Net Weighting Approach for Performance Optimization in Circuit Placement”, *Proc. ACM/IEEE Design Autom. Conf.*, 1991, pp. 620-625.

[32] R. S. Tsay, E. S. Kuh and C. P. Hsu, “PROUD: A Sea-of-Gates Placement Algorithm”, *IEEE Design & Test of Computers*, vol. 5(6), 1988, pp. 44-56.

[33] L.P.P.P. Van Ginneken,”Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay,” In *Proc. IEEE Intl. Symp. Circuits and Systems*, 1990, pp. 865–868.

[34] P. Villarrubia, “Important Considerations for Modern VLSI Chips”, *Proc. ACM/IEEE Intl. Symp. Physical Design*, 2003, pp. 1-6.

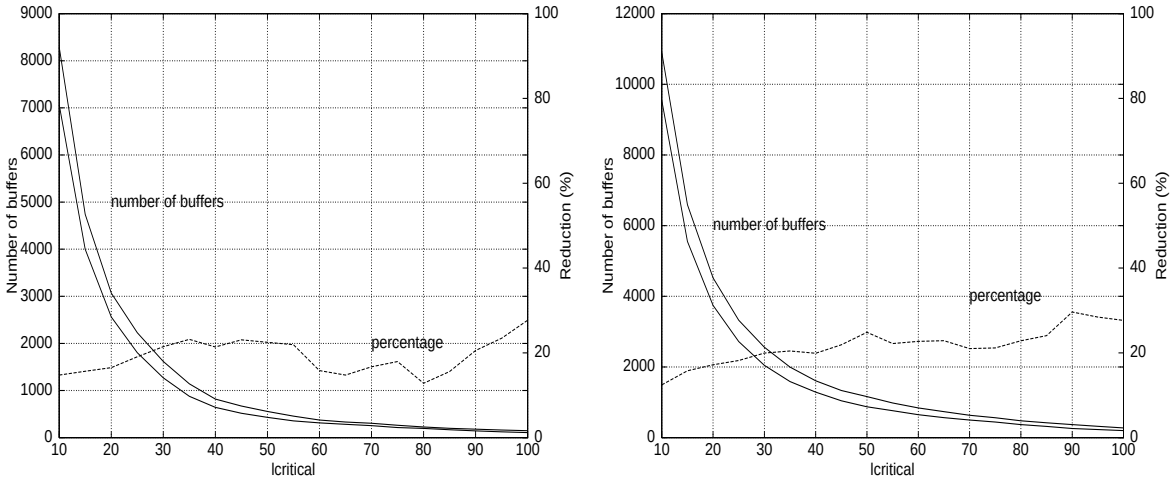
[35] J. Vygen, “Algorithms for Large-Scale Flat Placement”, *Proc. ACM/IEEE Design Autom. Conf.*, 1997, pp. 746-751.

[36] M. Wang, X. Yang and M. Sarrafzadeh, “DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits”, *Proc. IEEE Intl. Conf. Comp.-Aided Design*, 2001, pp. 260-263.

benchmark		boosting factor β				
		1	2	3	4	5
Design A	1	97.989	98.319	98.407	98.362	98.464
	2	1.132	0.920	0.829	0.998	0.887
	3	0.392	0.348	0.370	0.247	0.291
	4	0.151	0.145	0.145	0.142	0.115
	5	0.123	0.067	0.051	0.052	0.041
	6	0.015	0.004	0.002	0.003	0.007
	7	0.004	0.002	0.002	0.002	0.002
	8	0.001	0.001	0.001	0.001	0.001
	9	0.000	0.000	0.000	0.000	0.000
	10	0.000	0.000	0.000	0.000	0.000
	HPWL	1.000	1.046	1.118	1.183	1.205
Design B	1	92.627	93.046	93.397	93.229	93.145
	2	4.184	4.509	4.415	4.656	4.576
	3	1.633	1.365	1.331	1.301	1.351
	4	0.740	0.538	0.433	0.420	0.510
	5	0.338	0.227	0.177	0.145	0.198
	6	0.219	0.126	0.064	0.074	0.052
	7	0.097	0.035	0.034	0.025	0.015
	8	0.020	0.009	0.006	0.007	0.008
	9	0.001	0.002	0.002	0.001	0.004
	10	0.000	0.000	0.000	0.000	0.000
	HPWL	1.000	1.017	1.052	1.101	1.149
Design C	1	89.862	90.369	90.478	90.347	89.947
	2	6.781	6.884	7.252	7.356	7.980
	3	2.052	1.872	1.502	1.544	1.382
	4	0.650	0.547	0.448	0.448	0.413
	5	0.414	0.211	0.242	0.253	0.215
	6	0.175	0.082	0.059	0.038	0.042
	7	0.048	0.022	0.006	0.002	0.008
	8	0.008	0.004	0.003	0.002	0.003
	9	0.008	0.007	0.008	0.008	0.008
	10	0.000	0.000	0.000	0.000	0.000
	HPWL	1.000	1.046	1.085	1.127	1.179
Design D	1	93.225	93.506	93.215	93.169	93.150
	2	2.967	2.779	3.070	3.171	3.164
	3	1.337	1.306	1.380	1.330	1.462
	4	0.611	0.671	0.582	0.611	0.553
	5	0.334	0.279	0.375	0.276	0.293
	6	0.298	0.274	0.207	0.252	0.202
	7	0.123	0.077	0.084	0.101	0.087
	8	0.063	0.067	0.048	0.050	0.048
	9	0.014	0.012	0.010	0.010	0.012
	10	0.000	0.000	0.000	0.000	0.000
	HPWL	1.000	1.067	1.085	1.147	1.165

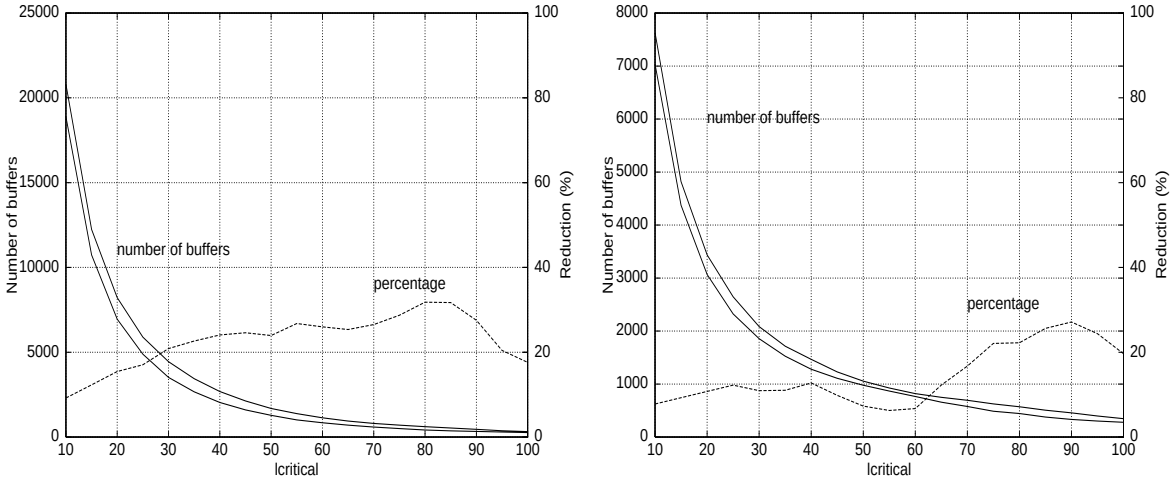
TABLE II

EFFECT OF BOOSTING ON THE WIRELENGTH DISTRIBUTION AND FINAL HPWL FOR VARIOUS BOOSTING FACTORS. RESULTS ARE AVERAGE OF FOUR SEEDS.



(a) Design A

(b) Design B



(c) Design C

(d) Design D

Fig. 8. Number of buffers that would be inserted in nets of the four benchmarks. We vary the critical length from 10% of the chip's half perimeter to 100% in steps of 5% and calculate the average number of buffers that would be inserted for both boosted and non-boosted placement. We also calculate the percentage improvement in number of buffers due to boosting. Boosting is consistently reducing the number of buffers.

benchmark	mode	Critical length percentage $l_{critical}$									
		10	20	30	40	50	60	70	80	90	100
Desgin A	regular	8292	3069	1617	818	556	373	305	226	180	148
	boost	7068	2562	1270	643	431	314	254	197	143	107
	improv	14.8%	16.5%	21.5%	21.4%	22.5%	15.8%	16.7%	12.8%	20.6%	27.7%
Design B	regular	10908	4522	2557	1608	1163	841	633	477	368	275
	boost	9550	3745	2047	1288	874	650	500	368	259	199
	improv	12.4%	17.2%	19.9%	19.9%	24.9%	22.7%	21.0%	22.9%	29.6%	27.6%
Design C	regular	20804	8222	4440	2682	1683	1135	803	613	454	318
	boost	18888	6951	3514	2036	1280	840	590	418	329	262
	improv	9.2%	15.5%	20.9%	24.1%	23.9%	26.0%	26.5%	31.8%	27.5%	17.6%
Design D	regular	7631	3437	2085	1466	1054	820	693	571	457	346
	boost	7036	3067	1857	1279	977	765	577	444	333	278
	improv	7.8%	10.7%	10.9%	12.7%	7.3%	6.7%	16.7%	22.2%	27.1%	19.6%

TABLE III

PERCENTAGE REDUCTION IN NUMBER OF BUFFERS FOR THE DIFFERENT CRITICAL LENGTHS FOR INDUSTRIAL BENCHMARKS.

benchmark	mode	Critical length percentage $l_{critical}$										HPWL
		10	20	30	40	50	60	70	80	90	100	
ibm01	Capo	5475	1970	1002	555	334	182	116	61	30	18	5.63
	Capo + Boost	5330	1875	868	500	277	161	96	50	30	18	5.83
	FengShui	5074	1853	912	531	293	184	100	53	29	17	5.17
	Dragon	4948	1752	868	474	271	161	92	45	23	11	5.15
ibm02	Capo	16880	6753	3507	2006	1181	738	489	342	249	190	16.0
	Capo + Boost	16188	6295	3169	1724	1025	635	431	287	200	156	16.4
	FengShui	16134	6468	3381	1851	1036	643	416	299	231	183	14.7
	Dragon	15258	5957	3016	1625	942	590	408	311	237	191	14.1
ibm07	Capo	20898	7799	3896	2148	1375	862	540	342	208	136	37.4
	Capo + Boost	19254	6821	3178	1658	958	611	381	229	111	63	38.1
	FengShui	17844	6375	2921	1552	855	536	367	231	115	54	32.8
	Dragon	17960	6382	3013	1403	755	463	293	139	77	40	33.1
ibm08	Capo	24343	9766	5316	3270	2174	1512	1074	768	552	374	39.3
	Capo + Boost	21660	8170	4259	2414	1562	1049	705	445	311	221	39.0
	FengShui	21734	8586	4511	2719	1716	1118	780	531	360	259	34.7
	Dragon	21824	8667	4609	2800	1793	1220	869	614	432	301	33.2
ibm09	Capo	16585	5817	2749	1467	766	387	235	124	48	24	33.4
	Capo + Boost	15341	5114	2221	1115	598	325	162	66	27	15	33.8
	FengShui	15351	5287	2337	1141	585	253	119	60	28	14	30.2
	Dragon	14561	4872	2069	987	468	205	87	41	21	11	29.8
ibm10	Capo	23585	7882	3546	1781	896	497	292	195	85	40	63.7
	Capo + Boost	22315	7009	3029	1405	690	351	188	96	50	32	63.8
	FengShui	21287	6988	3028	1533	795	422	248	157	114	73	56.9
	Dragon	20814	6634	2830	1412	716	394	230	128	65	43	56.6

TABLE IV

PERCENTAGE REDUCTION IN NUMBER OF BUFFERS FOR THE DIFFERENT CRITICAL LENGTHS FOR IBM (VERSION 2) EASY BENCHMARKS.

benchmark	mode	Total cut value per placement level									
		1	2	3	4	5	6	7	8	9	10
ibm01	Capo	91	339	672	1115	1587	2254	2984	3857	4798	5874
	Capo + Boost	91	334	685	1232	1834	2591	3443	4505	5545	6722
ibm02	Capo	229	626	1995	3574	4879	6094	7258	8264	9295	10272
	Capo + Boost	230	559	1783	3521	5520	7016	8577	10053	11235	12416
ibm07	Capo	832	2038	3325	5096	6999	9066	11175	13340	15641	18204
	Capo + Boost	849	2095	3553	5543	7872	10863	13703	16440	19051	21855
ibm08	Capo	826	1990	3184	4834	6502	8750	10924	13403	15821	18426
	Capo + Boost	828	2227	3745	5822	7816	10656	13645	16647	19407	22252
ibm09	Capo	444	1455	2427	3374	4888	6767	9208	11730	14607	17378
	Capo + Boost	437	1450	2467	3690	5548	8037	11027	14058	17085	20152
ibm10	Capo	785	1822	3335	5718	8107	11392	14720	18438	22232	25872
	Capo + Boost	730	1996	3493	6113	8962	13164	17552	22201	26068	29985

TABLE V

TOTAL NET CUT PER PLACEMENT LEVEL FOR IBM (VERSION 2) EASY BENCHMARKS.

benchmark	**flow**		**HPWL**	**VIOL-**	**time**	**Wire**	**SLACK**	**TNS**
	tool	mode		ATIONS	(s)	**length**	**(ns)**	**(ns)**
A	**indust**	NTD	3511326	0	771	4242633	-5.411	2969.5
		TD	3442277	0	1249	4206754	-5.208	3243.8
	CAPO	regular	3165932	0	189	4076119	-5.315	3178.3
		boost	3282640	0	191	4182982	-5.363	3083.6
B	**indust**	NTD	2826832	38	325	3491503	-0.660	28.107
		TD	2892279	52	602	3570024	-0.368	12.022
	CAPO	regular	2702130	82	183	3335483	-0.607	23.36
		boost	2758502	172	197	3260184	-0.342	5.107
C	**indust**	NTD	6874100	0	803	8374484	+0.263	0
		TD	7199963	0	1243	915298	+0.290	0
	CAPO	regular	6654505	107	227	8375939	-10.940	292.406
		boost	6978647	168	244	8711281	-4.389	113.042
D	**indust**	NTD	594392	5	278	846783	-1.882	1859.9
		TD	577484	5	511	836501	-1.880	1788.7
	CAPO	regular	590833	9	47	835962	-1.875	1810.6
		boost	629473	8	49	872089	-1.878	1799.5

TABLE VI

BENCHMARK RESULTS (AVERAGE OF FOUR SEEDS). **NTD** IS THE INDUSTRIAL PLACER IN A NON-TIMING DRIVEN MODE. **TD** IS THE INDUSTRIAL PLACER IN TIMING-DRIVEN MODE. IN **CAPO** FLOWS, **regular** IS FOR UNMODIFIED CAPO AND **boost** IS FOR CAPO IN BOOSTED MODE. **HPWL** IS HALF-PERIMETER WIRELENGTH. **VIOLATIONS** IS THE NUMBER OF DETAILED ROUTING VIOLATIONS (NUMBER OF NETS THAT ARE NOT COMPLETELY ROUTED SUCCESSFULLY). **SLACK PRE** IS THE NEGATIVE SLACK CALCULATED BEFORE ACTUAL ROUTING. **SLACK** IS THE NEGATIVE SLACK CALCULATED AFTER ROUTING. **TNS** IS THE TOTAL NEGATIVE SLACK.