Hierarchical Whitespace Allocation in Top-Down Placement

Andrew E. Caldwell, *Member, IEEE*,
Andrew B. Kahng, *Senior Member, IEEE*, and
Igor L. Markov, *Member, IEEE*

***Abstract*—Increased transistor density in modern commercial ICs typically originates in new manufacturing and defect prevention technologies [15], [16]. Additionally, better utilization of such *low-level* transistor density may result from improved software that makes fewer assumptions about physical layout in order to reliably automate the design process. In particular, recent layouts tend to have large amounts of whitespace, which is not handled properly by older tools. We observe that a major computational difficulty arises in partitioning-driven top-down placement when regions of a chip lack whitespace. This tightens balance constraints for min-cut partitioning and hampers move-based local-search heuristics such as Fiduccia–Mattheyses. However, the local lack of whitespace is often caused by very unbalanced distribution of whitespace during previous partitioning, and this concern is emphasized in chips with large overall whitespace.**

This paper focuses on accurate computation of tolerances to ensure smooth operation of common move-based iterative partitioners, while avoiding cell overlaps. We propose a mathematical model of hierarchical whitespace allocation in placement, which results in a simple computation of partitioning tolerance purely from relative whitespace in the block and the number of rows in the block. Partitioning tolerance slowly increases as the placer descends to lower levels, and relative whitespace in all blocks is limited from below (unless partitioners return "illegal" solutions), thus preventing cell overlaps. This facilitates good use of whitespace when it is scarce and prevents very dense regions when large amounts of whitespace are available.

Our approach improves the use of the available whitespace during global placement, thus leading to smaller whitespace requirements. Existing techniques, particularly those based on simulated annealing [21], [10], can be applied after global placement to bias whitespace with respect to particular concerns, such as routing congestion, heat dissipation, crosstalk noise and DSM yield improvement.

***Index Terms*—Algorithms, design automation, integrated circuit layout.**

I. Introduction

THE progression of Moore's law [18], [15] for commercial ICs has been so far maintained by steady increases in device densities as a result of innovations in manufacturing and defect prevention technologies [16]. At the same time, device density for a given process generation is also limited by the capabilities of EDA software and the assumptions made by software developers.

Historically, utilization rates increased (i.e., whitespace decreased) steadily with the introduction of three-layer, four-layer, and even five-layer metal technologies. In contrast with the preceding two-layer metal regime, three or more layers of metal brought the following changes: 1) the need for routing channels disappeared; 2) double-back (shared power and ground rail) standard-cell styles removed all whitespace between cell rows; and 3) built-in cell library porosity became the proxy for previous techniques such as explicit feedthrough insertion. By the late 1990s, standard-cell place-and-route (P&R) blocks exhibited a clear trend of decreasing whitespace, and instances with single-digit percentages of whitespace were not uncommon. On the other hand, as CMOS technology moved below quarter-micrometer feature sizes, utilization rates started to decrease (i.e., whitespace has *increased*) in 180- and 130-nm designs. One obvious reason for the increase in whitespace is the increasingly interconnect-limited nature of designs.

Interconnect limits arise due to pad-limited designs, and also due to conflicting goals (power and clock distribution, reliability, signal density, etc.) for the UDSM interconnect architecture. (For example, there is little point in pushing transistor density to its limits if the resulting cell library has inadequate porosity, or if power/ground distribution then uses up too much of the local routing resources.) Another reason for increased whitespace stems from the growing number of macro blocks in system-on-a-chip (SoC) designs which, even when floorplanned well, leave large layout areas in which to place the remaining standard-cell logic (e.g., "block limited" designs). However, increased whitespace is also a consequence of limitations in P&R tools. Guardbanding for these limitations—in particular, the ability to deal with routing congestion—in order to minimize the number of passes through the back-end tool flow is a major contributor to whitespace at the 180- and 130-nm nodes. Recent anecdotal evidence indicates that whitespace today varies considerably (from 20+% to around 70+%) with methodology and allowed design time.

Global top-down partitioning-based placers can be sensitive to the amount of whitespace, which contributes to tool limitations. As the *partitioning tolerance* decreases in the partitioning instance, the results of modern hypergraph partitioners (cutsize and the ability to find a solution that meets balance constraints) deteriorate significantly [8]. Partitioning solutions that violate prescribed tolerance often lead to cell overlaps when future subproblems run out of whitespace. Since nonuniform cell sizes generally worsen partitioner performance, modern cell libraries that have widely varying cell sizes exacerbate the difficulties of move-based partitioners. The role of partitioners in limiting the reduction of whitespace is the focus of our present research.

The essential components of a typical placement problem are the *placement region*, possibly with discrete allowed locations, the *cells* to be placed subject to various constraints, and the *netlist topology* that shapes the minimized objective function. Academic and commercial standard-cell placers often apply a top-down, divide-and-conquer approach to define an initial *global placement*. The top-down approach [20], [13], [2] decomposes the given placement problem into smaller problems by subdividing the placement region, assigning cells to subregions, reformulating constraints, and cutting the netlist—such that good solutions to subproblems combine into good solutions of the original problem. The concept of *placement blocks* is pivotal. A block represents: 1) a placement region with allowed locations; 2) a collection of cells to be placed in this region; 3) all nets incident to the cells; and 4) locations of all cells beyond the given region that are adjacent to the cells to be placed in the region; such external cells are considered to be terminals for the block, and their locations are fixed. A high-level pseudocode for top-down global placement in terms of placement blocks is shown in Fig. 1; sufficiently small partitioning instances are processed as *end-cases* by specialized partitioners or placers.

Manuscript received July 15, 2002; revised January 13, 2003.

A. E. Caldwell was with the University of California, Los Angeles, CA 90095 USA. He is now with Everychip Inc., Mountain View, CA 94041 USA (e-mail: andy@everychip.com).

A. B. Kahng was with the University of California, Los Angeles, CA 90095 USA. He is now with the Departments of Computer Science and Engineering and Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

I. L. Markov was with the University of California, Los Angeles, CA 90095 USA. He is now with the Department of Electrical Engineering and Computer Science, Univ. of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: imarkov@umich.edu).

Digital Object Identifier 10.1109/TCAD.2003.818375

```
Variables:      A queue of blocks
Initialization: A single block represents the original placement problem
Algorithm:      while (queue not empty)
                     dequeue a block
                     if (small enough) consider endcase
                     else
                          bipartition into smaller blocks
                          enqueue each block
```

Fig. 1. High-level outline of the top-down partitioning-based placement process.

Every block yields a hypergraph partitioning instance: nodes correspond to cells inside the block as well as propagated external terminals [7], and hyperedges are induced over the node set from the original netlist. In practice blocks are split through balanced min-cut hypergraph bisection with FM-type move-based heuristics [12], [9]; the performance of such heuristics on larger instances is improved through the multilevel paradigm [3], [11]. Global placement solutions place all cells near legal sites in cell rows with minimal overlaps. Detailed placement refinement then makes small perturbations to the placement to legalize (remove overlaps) or improve routability. The more overlapping cells and unroutable hotspots the detailed placer must address, the larger the risk of solution quality degradation. As more constraints or optimization objectives are added, e.g., timing, density, or *IR* drop, legalizing an overlapping or unroutable global placement while preserving solution quality becomes increasingly difficult. In many cases, the problem may not be fixable at the detailed placement stage. To this end, it is reasonable to aim for nonoverlapping, routable global placements. One common technique for managing both legality and routability during placement is controlling the distribution of whitespace—inserting more whitespace in congested areas. In this work, we focus on mathematical methods of whitespace management during top-down placement with the goal of producing nonoverlapping, routable global placements.

Hypergraph bisection instances arising in the top-down placement process in some cases have tight *balance constraints* [8], i.e., the sizes of partitions in the solution should not deviate from *target* partition sizes by more than prescribed amounts.[1] Such constraints may arise in several circumstances. In leading-edge microprocessors, the proportion of deliberately introduced free sites, i.e., whitespace, is limited.[2] In other designs, whitespace may be distributed unevenly, and thus some regions may be quite dense. To avoid overcapacity blocks in such regions, total cell area assigned to a block must closely match the area available for cells. Indeed, illegal solutions or excessively relaxed balance tolerances lead to uneven area utilization (i.e., proportion of whitespace in a given subblock) and, eventually, overlapping cells. Even when a legal solution is obtained by the partitioner, as the partitioner exploits its available tolerance one child of the partitioned block can have relatively less whitespace than its parent, so that partitioning constraints become tighter on lower levels of top-down placement, *even if average whitespace is large*. Such deviations in available whitespace cannot be easily corrected, essentially because cuts parallel to rows cannot be adjusted after partitioning. On the other hand, attempting to maintain available whitespace by imposing unnecessarily tight balance constraints will hurt solution quality and lead to increased wirelength in the layout.[3]

In this paper we develop a mathematical model to find better tolerances: not too big, to generally avoid overcapacity or unroutable blocks, and not too small, to facilitate common move-based partitioners. *The primary contribution of this paper is a new whitespace management framework for global placement whose utility is supported by experimental evidence. The framework can include consideration of layout density issues, such as routability and IR-drop, during the top-down placement process*.

Whitespace allocation can be further improved using recent annealer-based algorithms for detailed placement that address routing congestion [21], [10], heat dissipation, crosstalk noise, DSM yield improvement, etc. Modifications to global placement have been proposed targeting wirelength [4] and congestion [17]. We believe that our mostly mathematical contribution, being an easy addition to top-down partitioning-based placers, provides a common denominator for such work.

The remaining text is organized as follows. Basic notation is introduced in Section II and straightforward facts about whitespace are mentioned. In Section III, we control splitting of single blocks by maximizing partitioning tolerance for given *whitespace deterioration*. This process is described mathematically. Most surprisingly, *relative whitespace* and *relative tolerances* are connected independently of whitespace deterioration, site and cell areas. This leads to a simple computation of bipartitioner parameters in terms of relative whitespace in the block and the number of rows in the block. Experimental validation is presented in Section IV. Section V concludes with directions for ongoing work.

II. Whitespace Fundamentals

Let the block have *site area* S, cell area C (typically with $C \leq S$), *absolute whitespace* $W = \max\{S - C, 0\}$, and *relative whitespace*

[1]See [1] for a review of netlist partitioning formulations and constraints.

[2]Deliberate introduction of whitespace may be related to pin-limited designs, limits of power distribution/dissipation density, and the need to maintain autoroutability. While these phenomena are increasingly addressed by packaging and process technology, better architecture and circuit design techniques, multilayer interconnect processes, and cell library design, block-limited designs are becoming increasingly common. This accounts for the large range of whitespace currently seen in SoC designs.

[3]The work of [8] in particular showed that iterative move-based partitioners perform poorly with small tolerance. The authors of [8] proposed the technique of *intermediate relaxations* to trade CPU time for solution quality in such circumstances. Their work is orthogonal to ours: we address the question of how to best control the allocation of whitespace (via bounds on partition sizes) during the top-down partitioning process.

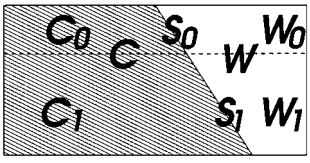

Fig. 2. Basic variables for a block and two child blocks: site area S, cell area C, and whitespace $W = S - C$.

$w = W/S$.[4] A geometric bipartitioning of the block entails *site areas* S_0 and S_1 in child blocks, such that $S_0 + S_1 = S, S_0 \leq S$, and $S_1 \leq S$. Any hypergraph bipartitioning solution implies *cell areas* C_0 and C_1 in child blocks, such that $C_0 + C_1 = C, 0 \leq C_0, 0 \leq C_1$ (if in the original block $C \leq S$, we will also require $C_0 \leq S_0, C_1 \leq S_1$). The input to a hypergraph bipartitioner must specify both the netlist and the allowed ranges for C_0 and C_1, i.e., bounds $C_0^{\min} \leq C_0 \leq C_0^{\max}, C_1^{\min} \leq C_1 \leq C_1^{\max}$ (with $C_0^{\max} + C_1^{\max} > C$ possible). These bounds establish *absolute* $T_j = C_j^{\max} - C_j^{\min}$ and *relative* $\tau_j = T_j/C$ partitioning tolerances for $j = 0, 1$ ($\tau_0 = \tau_1$ not required, but often holds). *Absolute* and relative whitespace in child blocks are defined by $W_j = S_j - C_j$ and $w_j = (W_j/S_j)$ (see Fig. 2).

In order to express w in terms of w_0 and w_1, write $w = (S - C/S) = (S_0 - C_0/S) + (S_1 - C_1/S) = (S_0/S)(W_0/S_0) + (S_1/S)(W_1/S_1)$, hence

$$w = \frac{S_0}{S} w_0 + \frac{S_1}{S} w_1. \tag{1}$$

Since $(S_1/S) = 1 - (S_0/S)$, relative whitespace in a block is a convex combination of relative whitespace in child blocks. Subsequently relative whitespace in a block is never smaller than that in every child block. If one child block has more relative whitespace than its parent, then the other child block has less. If we wish to limit relative whitespace in blocks from below, it suffices to consider end-case blocks only. In the special case when child blocks have equal site area, (1) contains an arithmetic average. In practice S_0/S can be small, e.g., 1/3 when a three-row block is bipartitioned horizontally, and even smaller than $1/3$ when some sites in the one-row child block are obstructed by special wiring.

Given a collection of nonoverlapping placement blocks that cover the layout, e.g., in the course of top-down placement, one can recursively apply (1) to show that the *average relative whitespace* for the design (i.e., relative whitespace for the top-level block) is a convex combination of relative whitespace in individual blocks.

Overcapacity cell area (or *overfill*) in a block is defined by $\Theta = \max\{0, C - S\}$, *relative overfill*—by $\theta = \Theta/S$. Given a collection of nonoverlapping blocks that cover the layout, the *total overfill* Θ^0 for the design is the sum of overfill in all blocks; *average relative overfill* θ^0 for the design is computed by dividing Θ^0 by the site area in all blocks. Similar to average relative whitespace, average relative overfill is a convex combination of relative overfills in individual blocks.

For a given block, the relative whitespace and relative overfill cannot be simultaneously bigger than zero, and ensuring nonzero whitespace in all blocks precludes overfill. Assuming nonzero relative whitespace (at the top level), we will require that for each block split the relative whitespace in each child block is at least αw, where w is the relative whitespace in the block and $0 \leq \alpha \leq 1$ is *whitespace deterioration*, i.e.

$$w_0 \geq \alpha w \quad \text{and} \quad w_1 \geq \alpha w. \tag{2}$$

For practical purposes, $\alpha = 1$ and may be overly restrictive, as it entails partitioning with zero tolerance in the proportion of site area, while $\alpha = 0$ may be too loose, as it allows for child blocks with no whitespace. As α approaches one, the distribution of whitespace in the final placement approaches uniform. An α of zero allows for fully utilized regions of the layout. This can improve wirelength and timing, but may also result in routability problems. It is desirable, then, to adjust α on a per-block basis to account for maximum allowed layout densities in leaf-level blocks. For example, in areas of high predicted routing density, α should be high, whereas in sparser areas it can safely be set to a low value α, allowing for denser placement but still ensuring a nonoverlapping result.

The following is the key observation for hierarchical whitespace allocation in top-down placement.

Theorem: Assuming nonzero average relative whitespace w_0 in the design, the result of top-down placement will have zero total overfill if every block split is performed with whitespace deterioration $\alpha_i > 0$, possibly different for every block.

Proof: Relative whitespace in every end-case block will then be at least $\alpha_0 \ldots \alpha_n w_0 > 0$.

In partitioning-driven top-down placement, whitespace deterioration is controlled with balance tolerance that constraints partitioning solutions. □

Corollary: Top-down placement results in a zero-overfill placement if balance tolerances correspond to strictly positive whitespace deterioration and all partitioning solutions are legal.

Excessively tight (i.e., to close to one) whitespace deterioration may allow no legal solutions, e.g., for purely number partitioning reasons if it entails tolerance below the size of one site.[5] Small tolerance can also imply poorer partitioning quality because it restricts the solution space and may incapacitate move-based partitioners by disallowing moves of large cells (when their sizes are bigger than tolerance).

On the other hand, [5] empirically shows that higher partitioning tolerances result in small placement wirelength. Therefore, it is important to keep whitespace high enough to prevent overcapacity blocks and, at the same time, keep partitioning tolerances high to ensure small cuts and, thus, wirelength.

III. Whitespace Allocation

In this section, we will control block splits by maximizing partitioning tolerance for given whitespace deterioration; this results in a number of useful properties: (6), (7), and (8). Most surprisingly, relative whitespace and relative tolerances are recursively connected (12) independent of whitespace deterioration. This allows to compute optimal relative tolerance given the initial and final relative whitespace. The latter can be computed solely from the initial relative whitespace (10).

We show how to determine $C_j^{\min}$ and $C_j^{\max}$ for a block only from its relative whitespace and the number of rows covered (16). Our formulas, derived to account for the worst case possible, can be transparently adjusted for optimism, which may be useful in designs with abundant whitespace.

A. Splitting a Block

We compute $C_j^{\max}$ and $C_j^{\min}$ for child blocks of a particular block assuming given α. The resulting tolerances for both partitions appear equal; thus, the relative tolerance τ is determined by α. It is somewhat surprising that α, w, and τ are connected independently of S_j or C_j and any two of them imply a particular value for the third, which can be computed by a simple formula. These relations facilitate further modeling of recursive block splits in the next subsection.

[4]We will sometimes refer to W as *whitespace*.

[5]Cell sizes are typically small integer multiples of site size.

Rewrite (2) as

$$\alpha\frac{S-C}{S} \le \frac{S_j - C_j}{S_j} \Rightarrow C_j S \le (1-\alpha)S_j S + \alpha S_j C$$
$$\Rightarrow C_j \le (1-\alpha)S_j + \alpha\frac{S_j}{S}C$$

adding the original bounds for C_j, we get

$$0 \le C_j \le \min\left\{C, S_j, (1-\alpha)S_j + \alpha\frac{C}{S}S_j\right\}.$$

Since $0 \le \alpha \le 1$ and $0 \le C \le S$, we have

$$(1-\alpha)S_j + \alpha\frac{C}{S}S_j \le (1-\alpha)S_j + \alpha S_j \le S_j.$$

The above is now simplified[6]

$$0 \le C_0 \le \min\left\{C, (1-\alpha)S_0 + \alpha\frac{C}{S}S_0\right\} =: C_0^{\max} \quad (3)$$
$$0 \le C_1 \le \min\left\{C, (1-\alpha)S_1 + \alpha\frac{C}{S}S_1\right\} =: C_1^{\max}. \quad (4)$$

The remaining constraint $C_0 + C_1 = C$ is now equivalent to

$$C_0 \ge \max\{0, C - C_0^{\max}\} =: C_0^{\min},$$
$$C_1 \ge \max\{0, C - C_1^{\max}\} =: C_1^{\min}. \quad (5)$$

When C is very small compared to S (i.e., the block has a lot of whitespace) and α sufficiently small, $C_j^{\max}$, and $C_j^{\min}$ may degenerate into C and zero, respectively. In such cases, all cells are allowed to go into one partition and no further analysis is required until a child block appears with small enough white space to produce nontrivial partitioning tolerance.

In the following analysis, we assume that all cells can never go into one partition (worst case); therefore,[7] $C_j^{\max} = (1-\alpha)S_j + \alpha(S_j/S)C$ and $C_j^{\min} = C - C_{1-j}^{\max}$. Now $T_j = C_j^{\max} - C_j^{\min} = C_0^{\max} + C_1^{\max} - C = (1-\alpha)(S-C)$, i.e., absolute tolerances for partitions are equal. Furthermore, $\tau = T/C = (1-\alpha)(S/C - 1)$ from which straightforward calculations lead to

$$\tau = \frac{(1-\alpha)w}{1-w} \quad (6)$$
$$\alpha = (\tau+1) - \frac{\tau}{w} \quad (7)$$
$$w = \frac{\tau}{\tau+1-\alpha}. \quad (8)$$

B. Hierarchical Whitespace Allocation

It has been shown above that, for a given block, feasible ranges for partition capacities are uniquely determined by α. This section discusses methods of determining values of α. The methods presented do not account for routability, as this will depend heavily on the capabilities of the specific router to be used. The framework presented can be, however, simply extended to utilize congestion estimation by increasing or decreasing the target whitespace of the leaf-level blocks or the optimism applied to α. Further, we assume layout cut lines are shifted to match w_0 and w_1 as closely as possible to w in an attempt to recover the whitespace degradation. This allows future partitioning steps the benefits of high tolerance, but may be inappropriate for low-utilization designs, as it attempts to distribute whitespace uniformly, resulting in an overly spread placement.

We start with the top-level block having w_0 whitespace and go on to further blocks with whitespace

$$w_{i+1} = \alpha_i w_i. \quad (9)$$

We distinguish between blocks being split by cut lines parallel to rows and those perpendicular to rows. This is because perpendicular cut lines can be adjusted after partitioning to achieve nearly any desired distribution of whitespace among child blocks, which has the effect of $\alpha \approx 1$. In other words, we only have to consider row-parallel cut lines. Let R be the number of rows; the expected number n of *recursively applied parallel block splits* will be $n = \lceil \log_2 R \rceil$, assuming that rows are distributed evenly between child blocks at every block split.

To prevent overcapacity blocks and improve routability, assume also that whitespace in every block below the current block must be at least $\bar{w} (\le w_0)$. We find $\bar{w}$, observing that end-case blocks have $\alpha = 0$, since they are not partitioned further. Thus, from (8) we get

$$\bar{w} = w_n = \frac{\bar{\tau}}{\bar{\tau}+1}. \quad (10)$$

Note that $\bar{w}$ can be determined individually for each block being partitioned, and provides a method of accounting for routability and similar density-related issues. A straightforward way to determine α_i is to assume that they are all equal. This leads to[8]

$$\alpha = \sqrt[n]{\frac{\bar{w}}{w_0}} = \sqrt[n]{\frac{\bar{\tau}}{w_0(\bar{\tau}+1)}}. \quad (11)$$

However, assuming all τ_i equal appears more practical, since balanced partitioners typically require a certain relative tolerance to be successful regardless of whitespace deterioration. In a different, improved approach, we combine (7) and (9) to get rid of α

$$w_{i+1} = (\tau_i + 1)w_i - \tau_i. \quad (12)$$

Now, assuming that all τ_i equal yields $w_n = (\tau+1)^n w_0 - \tau(\tau+1)^{n-1} - \cdots - \tau = (\tau+1)^n w_0 - \tau((\tau+1)^n - 1)/((\tau+1)-1)$, resulting in

$$\bar{w} = w_n = (\tau+1)^n w_0 - (\tau+1)^n + 1$$

and

$$(\tau+1)^n = \frac{1-\bar{w}}{1-w_0}. \quad (13)$$

Therefore, we can replace the straightforward (11) with

$$\tau = \sqrt[n]{\frac{1-\bar{w}}{1-w_0}} - 1 \quad (14)$$

and combine (13) and (10) with $\tau_n = \tau$ to find a closed expression for τ

$$\tau = \frac{1}{\sqrt[n+1]{1-w_0}} - 1. \quad (15)$$

Finally, (15), and (7) give a closed expression for whitespace deterioration α in terms of relative whitespace w in the current block and the number R of rows in the block

$$\alpha = \frac{\sqrt[n+1]{1-w} - (1-w)}{w\sqrt[n+1]{1-w}}, \quad n = \lceil \log_2 R \rceil. \quad (16)$$

[6]Here we also define $C_0^{\max}$ and $C_1^{\max}$, while $C_0^{\min}$, and $C_1^{\min}$ are defined in (5).

[7]Our analyses hold for blocks allowing all cells in one partition; however, the proposed values of $C_j^{\max}$ on lower levels are not the best possible. In a way, we assume the pessimistic case.

[8]Due to essentially uncontrollable distribution of whitespace in child blocks and to compensate for small whitespace fluctuations after perpendicular block splits, α_i can be recomputed for every block, given the actual amount of whitespace in the block.

Values $C_j^{\max}$ and $C_j^{\min}(j = 0, 1)$ supplied to partitioner can be directly computed by (3), (4), and (5).

This computation of $C_j^{\max}$ and $C_j^{\min}$ can be performed for every block using the exact amount of whitespace available after partitioning.[9] While resulting tolerances are likely to increase toward lower levels, they will preserve original lower bounds for relative whitespace in every block[10] and thus prevent overcapacity blocks. Compared to the pessimistic *a priori* tolerances, such increased tolerances can result in cut improvements during hypergraph partitioning and cause smaller total placement wirelength.

C. Splitting Overcapacity Blocks

While our approach guarantees no overcapacity blocks and no cell overlaps under certain conditions, these conditions may not always hold. In particular, a partitioner may return illegal solutions when there are cells larger than partitioning tolerance or, more generally, when balanced solutions do not exist for number partitioning reasons. This typically happens when partitioning small blocks with tight tolerances, where each cell accounts for a considerable percent of the block's total cell area.

When splitting such blocks, we will minimize the maximal relative overfill in child blocks, which is equivalent to equal relative overfill according to Section II.[11] Hence, the desired partition capacities C_1 and C_2 need to satisfy $\theta^0 = (C_0 - S_0/S_0) = (C_1 - S_1/S_1) = \theta^1$ and $C_0 + C_1 = C$; they can be computed via

$$C_0 = \frac{S_0}{S}C, \quad C_1 = \frac{S_1}{S}C. \tag{17}$$

These considerations do not allow for nonzero partitioning tolerance, since even the slightest imbalance in the resulting partitioning solution would lead to an increase in relative overfill. However, there are currently no practical algorithms available for zero-tolerance partitioning. Therefore, it is necessary to artificially introduce partitioning tolerance. Appropriate values should be chosen corresponding to the cell area distribution and the capabilities of the partitioning algorithm used, e.g., we used the smaller of 2% and the overall whitespace in the design. Equation (17) is used for computing target partitioning capacities.

IV. Experimental Validation

In this section we present a comparison of our proposed method with a simpler alternative whitespace allocation strategy for horizontal cuts. In this simpler method, the target partition balances are proportional to site areas in the resulting partitions and horizontal cut tolerances are constant.

A. Top-Down Placement Testbench

We have implemented a full-fledged global placer that reads standard-cell row-based designs from Cadence Design Systems, Inc., in library exchange format and design exchange format (LEF/DEF) and produces cell placements with little or no overlap. Placement blocks are split via min-cut partitioning, which is implemented as a variant of multilevel Fiduccia–Mattheyses [9] for blocks with 200 cells or more and (flat) Fiduccia–Mattheyses for smaller blocks. The placer chooses vertical or horizontal block splits depending on the blocks' aspect ratio to always cut along the longest side of a block. When partitioning is performed with vertical cut line, the current block is bisected by a straight line and sites in the two resulting regions are counted to produce the site area in each region. The target partition capacities (cell areas) are then computed to be proportional to the site areas and add up to the cell area in the block. Vertical partitioning is performed with 10% tolerance in all our experiments. After partitioning, when the actual total cell area in each partition is available, the vertical straight-line determining block boundaries is optimally shifted to equalize relative whitespace in the blocks.

We use two different methods to allocate whitespace for blocks split by horizontal cut lines. In one set of experiments, we simply used the same computations as for the vertical block splits with partitioning tolerance being 2%, except for the inability to adjust horizontal cut lines after partitioning.[12] The second method follows (16), (3), (4), and (5). All overcapacity blocks are treated as explained in Section III-C.

We do not apply "legalization," "cycling," "overlapping" [14], or any other techniques that would move cells between existing blocks or change block boundaries other than during top-down block splits. Without such postprocessing the effects of different approaches to whitespace allocation come clear, and fair comparisons can be made. Moreover, even with such pessimistic view of cell overlaps, our top-down placements have very few cell overlaps. Our top-down placer is implemented in the `C++` language. Executables are compiled with the `SunPro CC4.2` compiler on Solaris2.6 and optimized with `-O5`. We use Sun Ultra-10 300-MHz workstations with 256 Mb of memory.

TABLE I
Statistics for the Testcases Used in Our Experiments

Test Case	Core cells	Pads	Nets	core rows	% whitespace
case1	2741	545	2842	24	11.29
case2	11471	662	11673	42	24.28
case3	12146	711	10880	53	23.40
case4	20392	185	21987	88	14.07
case5	85395	177	87272	301	30.15
case6	117440	177	101726	250	14.21

B. Results

We report experimental results for six industrial testcases, ranging from 2.7 K *cells* to more than 117 K *cells*, with whitespace from 11% to 30% (see Table I).[13] Relative average overfill for final placements is evaluated as explained in Section 2 and is given together with final wirelength and CPU time in Table II.

Our experiments show the following.

- Our proposed method allows for maximal use of available whitespace, producing placements with typically 10% smaller wirelength than the fixed 2% method.
- Increasing the fixed tolerance enough to achieve competitive wirelengths produces placements with more overlap than our proposed method.
- For every test case, our proposed method produces either lower wirelength and the same amount of relative overlap, or no relative overlap and lower wirelength, than the fixed 5% method.

[9]Compared to a possible *a priori* computation for all blocks that only assumes relative whitespace at the top level and the number of rows in the design, but assumes that *every block has the worst possible relative whitespace after partitioning*.

[10]In the assumption that all partitioning tolerances are satisfied.

[11]Since the relative overfill in the original block is a convex combination of the relative overfills in child blocks, one of the child blocks having smaller overfill implies the other having a higher overfill.

[12]Which explains our choice of smaller tolerance.

[13]Whitespace is measured by dividing the site area available to cells by the total cell area in the netlist. In this we accounted for sites put out of use by power stripes and other obstacles, but such adjustments turn out to be very small.

TABLE II
PLACER RUNTIME (CPU SECONDS ON A 300-MHz SUN ULTRA-10), HALF-PERIMETER WIRELENGTH OF FINAL PLACEMENTS AND AVERAGE RELATIVE OVERFILL **R/O**) IN PERCENT FOR OUR PROPOSED TOLERANCE COMPUTATION METHOD AND A STRAIGHTFORWARD METHOD WITH FIXED TOLERANCE. ALL NUMBERS ARE AVERAGES OF FIVE RUNS

	Proposed			Fixed Tolerance 5%			Fixed Tolerance 2%		
	HPWL	CPU	R/O	HPWL	CPU	R/O	HPWL	CPU	R/O
case1	5.85e5	28.5	0.00	5.95e5	27.4	0.00	6.32e6	28.1	0.00
case2	2.75e5	158.2	0.00	2.80e5	151.1	0.00	3.03e5	158.7	0.00
case3	2.50e5	140.5	0.00	2.61e5	140.3	0.00	2.89e5	153.8	0.00
case4	5.83e6	298.3	0.03	5.83e6	282.9	0.04	6.38e6	295.1	0.00
case5	2.26e7	1894	0.00	2.29e7	1802	0.00	2.45e7	1784	0.00
case6	1.16e8	2479	0.01	1.16e8	2438	0.08	1.25e8	2424	0.00

V. CONCLUSION AND FUTURE WORK

Our work addresses the global placement of modern ICs in which whitespace can vary dramatically across the layout region.[14] The proposed methods allow handling very small amounts of whitespace and can be used to appropriately distribute large amounts of whitespace when it is available. Our framework can be extended to account for target utilizations at particular areas of the layout region, allowing for consideration of density-related effects such as routability and *IR*-drop.

We have derived simple formulas for optimal[15] hierarchical whitespace allocation in top-down partitioning-driven placement of standard-cell row-based application-specified integrated circuit designs. With a straightforward practical adjustment, partitioning tolerance slowly increases as the placer descends to lower levels. We limit relative whitespace in all blocks from below, and this constraint prevents overcapacity blocks under the assumption that all partitioning solutions are legal. We point out that *using the proposed formula requires almost no additional programming and incurs practically no runtime penalty. The use of this formula also does not appear to conflict with popular placement algorithms and design practices*.

Experiments on industrial testcases with up to 114 K cells show that our technique practically achieves cell overlaps on the order of hundredths of a percent of the total areas and is superior to a commonly used straightforward technique, since it achieves better wirelength under the assumption of comparable cell overlaps (which can be provided by sufficiently small partitioning tolerance).

In real-world implementations, whitespace allocation can be further improved using recent annealer-based algorithms for detailed placement that address routing congestion [10], [21], heat dissipation, or other concerns. Modifications to global placement have been proposed targeting wirelength [4] and congestion [17]. We believe that our mostly mathematical contribution, being an easy addition to top-down partitioning-based placers, provides a common denominator for such work without any noticeable costs. This can be contrasted with nontrivial development required to implement algorithms based on simulated annealing and their runtime costs [21].

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration*, vol. 19, pp. 1–81, 1995.

[14]All implementations of this work are publicly available at http://vlsicad.cs.ucla.edu/software/PDtools/.

[15]In the worst case sense; also under the assumption of constant partitioning tolerance and legal partitioning solutions on all levels.

[2] C. J. Alpert, T. Chan, D. J.-H. Huang, I. Markov, and K. Yan, "Quadratic placement revisited," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 752–757.
[3] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *ACM/IEEE Design Automation Conf.*, 1997, pp. 530–533.
[4] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "The effect of free space in global placement," in *Proc. ICCAD 2002*, pp. 746–751.
[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Relaxed partitioning balance constraints in top-down placement," in *ASIC'98*, pp. 229–232.
[6] W. Deng, private communication.
[7] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 92–98, Jan. 1985.
[8] S. Dutt and H. Theny, "Partitioning around roadblocks: Tackling constraints with intermediate relaxations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, pp. 350–355.
[9] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[10] B. Hu and M. Marek-Sadowska, "Congestion minimization during placement without estimation," in *Proc. ICCAD 2002*, pp. 739–745.
[11] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526–529.
[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.
[13] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 356–365, Mar. 1991.
[14] D. J. Huang and A. B. Kahng, "Partitioning-based standard cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, 1997, pp. 18–25.
[15] D. Jensen, C. Gross, and D. Mehta. (1998, Jan.) New industry document explores defect reduction technology challenges. *Micro Magazine* [Online]. Available: http://www.micromagazine.com/archive/98/01/jensen.html
[16] D. Jensen and W. Fosnight. (1998, Oct.) Defect prevention and elimination: Where the rubber hits the road(map). *Micro Magazine* [Online]. Available: http://www.micromagazine.com/archive/98/10/jensen.html
[17] A. Rohe and U. Brenner, "An effective congestion driven placement framework," in *Proc. ISPD 2002*, pp. 6–11.
[18] "National Technology Roadmap for Semiconductors," Semiconductor Industry Association (SIA), San Jose, CA, 1997.
[19] H. D. Simon and S.-H. Teng, "How good is recursive bisection?," *SIAM J. Sci. Comput.*, vol. 18, no. 5, pp. 1436–1445, 1997.
[20] R. S. Tsay and E. Kuh, "A unified approach to partitioning and placement," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 521–633, May 1991.
[21] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability-driven whitespace allocation for fixed-die standard cell placement," in *Proc. ISPD 2002*, pp. 42–48.