Hierarchical Whitespace Allocation in Top-down Placement

Andrew Caldwell and Igor Markov

UCLA Computer Science Dept., Los Angeles, CA 90095-1596

{caldwell,imarkov}@cs.ucla.edu

Abstract

Increased transistor density in modern commercial ICs typically originates in new manufacturing and defect prevention technologies [13, 14]. 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. We observe that existing block- and chip-level layout instances can have considerable amounts of *whitespace* – i.e., *deliberately unused*, costly layout area – in order to reliably produce high-quality automatic layout solutions. Indeed, modern designs typically have from several percent up to 20% or more whitespace [5].

Whitespace can be the result of several phenomena, including pin-limited designs, limits of power distribution/dissipation density, and the need to maintain autoroutability. The first two of these phenomena are intrinsic to the design; the third is the result of limits in the place-and-route tools. A fourth cause of whitespace, also due to tool limitations, is the use of move-based hypergraph partitioning in top-down placement of standard-cell row-based designs. 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 [7]. Partitioning solutions that violate prescribed tolerance often lead to cell overlaps when future subproblems run

out of whitespace. Since non-uniform cell sizes generally worsen partitioner performance, modern cell libraries that have widely varying drive strengths and 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.

It is known that solution quality for low-tolerance partitioning can be improved for the cost of additional CPU time [7]. Furthermore, simply increasing the tolerance, e.g., when partitioning a subinstance perpendicular to cell rows (since there is flexibility in locating the resulting cutline), results in smaller placement wirelength [4]. This work focuses on accurate computation of tolerances to facilitate use 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. Our approach improves the use of the available whitespace, thus leading to smaller whitespace requirements.

1 Introduction

Increased device density in commercial ICs, commonly perceived as necessary to maintain the progression of Moore's law [15, 13], has been primarily seen as a result of innovations in manufacturing and defect prevention technologies [14]. At the same time, such device density for a given process generation is also limited by the capabilities of EDA software [18]. A particular limitation is caused by the *whitespace* that move-based physical design optimizations, in particular *placement*, require to reliably converge to low-cost solutions.

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 [17, 11, 1] 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 (i) a placement region with allowed locations, (ii) a collection of cells to be placed in this region, (iii) all nets incident to the cells, and (iv) 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 Figure 1; sufficiently small partitioning instances are processed as *end-cases* by specialized partitioners or placers.

Every block yields a hypergraph partitioning instance: nodes correspond to cells inside the block as well as propagated external terminals [6], 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 [10, 8]; The performance of such heuristics on larger instances is improved through the multilevel paradigm [2, 9]. Global placement solutions place all cells at legal sites in cell rows with no overlaps. Detailed placement refinement then makes small perturbations to exploit such degrees of freedom as cell mirroring, electric equivalence substitution, or shifting a cell slightly within its row to improve routability.

Hypergraph bisection instances arising in the placement process tend to have tight *balance constraints* [7], i.e., the sizes of partitions in the solution should not deviate from *target* partition sizes by more than prescribed amounts (see [3] for a review of netlist partitioning formulations and constraints). Such constraints arise because the proportion of deliberately introduced free sites (i.e., whitespace) in leading edge deep-submicron designs is limited.[1] To avoid overcapacity blocks, total cell area assigned to a block must always closely match the area available for cells. Illegal solutions or excessively relaxed balance tolerances lead to uneven area utilization (i.e., proportion of whitespace in a given sub-block) 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.[2]

In this work we develop a mathematical model to find better tolerances: not too small — to generally avoid overcapacity blocks, and not too big — to facilitate common move-based partitioners. *The primary contribution of this work is a new whitespace computation whose utility is supported by experimental evidence.*

The remaining text is organized as follows. Basic notation is introduced in Section 2 and straight-

[1] Deliberate introduction of whitespace traditionally stems from three phenomena, namely, pin-limited designs, limits of power distribution/dissipation density, and the need to maintain autoroutability. However, these phenomena are increasingly addressed by packaging and process technology, better architecture and circuit design techniques, multilayer interconnect processes, and cell library design. Thus, whitespace in modern block- and chip-level instances typically ranges from several % to 20% [5].

[2] The work of [7] in particular showed that iterative move-based partitioners perform poorly with small tolerance. The authors of [7] 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.

```
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
```

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

forward facts about whitespace are mentioned. In Section 3, 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 4. Section 5 concludes with directions for ongoing work.

2 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* $w = W/S$. A geometric bipartitioning of the block entails *site areas* S_0 and S_1 in child blocks ($S_0 + S_1 = S$, $S_0 \leq S$, $S_1 \leq S$). Any hypergraph bipartitioning solution implies *cell areas* C_0 and C_1 in child blocks ($C_1 + C_2 = C$, $0 \leq C_0$, $0 \leq C_1$; 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 $w_j = \frac{W_j}{S_j}$ (see Figure 2).

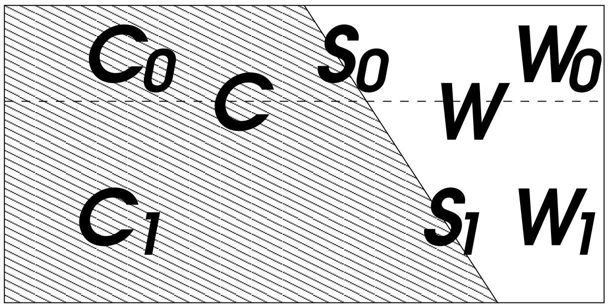

Figure 2: Basic variables for a block and two child blocks: site area S, cell area C and whitespace $W = S - C$.

In order to express w in terms of w_0 and w_1, write $w = \frac{S-C}{S} = \frac{S_0-C_0}{S} + \frac{S_1-C_1}{S} = \frac{S_0}{S}\frac{W_0}{S_0} + \frac{S_1}{S}\frac{W_1}{S_1}$, hence

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

Since $\frac{S_1}{S} = 1 - \frac{S_0}{S}$, 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, Equation (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 non-overlapping 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 non-overlapping 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. Similarly 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 can not be simultaneously bigger than zero, and ensuring non-zero whitespace in all blocks precludes overfill. Assuming non-zero average 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.

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

For practical purposes, $\alpha = 1$ 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. It is reasonable, however, to assume $\alpha \approx 0$ for end-case blocks and $\alpha \approx 1$ for blocks at high levels of large designs.

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

Theorem Assuming non-zero 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 $\alpha_0 \dots \alpha_n w_0 > 0$.

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 1) whitespace deterioration may allow no legal solutions, e.g., for purely number partitioning reasons if it entails tolerance below the size of one site.[3] 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, [4] 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.

3 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

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

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 formulae, derived to account for the worst case possible, can be transparently adjusted for optimism, which may be useful in designs with abundant whitespace.

3.1 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 and 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.

Rewrite (2) as

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

adding the original bounds for C_j we get

$$0 \leq C_j \leq \min\{C, S_j, (1-\alpha)S_j + \alpha\frac{C}{S}S_j\}$$

Since $0 \leq \alpha \leq 1$ and $0 \leq C \leq S$, we have $(1-\alpha)S_j + \alpha\frac{C}{S}S_j \leq (1-\alpha)S_j + \alpha S_j \leq S_j$.

The above is now simplified[4]

$$0 \le C_0 \le \min\{C, (1-\alpha)S_0 + \alpha \frac{C}{S} S_0\} =: C_0^{max} \tag{3}$$

$$0 \le C_1 \le \min\{C, (1-\alpha)S_1 + \alpha \frac{C}{S} S_1\} =: C_1^{max} \tag{4}$$

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

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

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 0 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 non-trivial partitioning tolerance.

In the following analysis, we assume that all cells can never go into one partition (worst case), therefore[5] $C_j^{max} = (1-\alpha)S_j + \alpha \frac{S_j}{S} C$ 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

$$\tau = \frac{(1-\alpha)w}{1-w} \tag{6}$$

$$\alpha = (\tau + 1) - \frac{\tau}{w} \tag{7}$$

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

[5]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.

$$w = \frac{\tau}{\tau + 1 - \alpha} \tag{8}$$

3.2 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.

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

$$w_{i+1} = \alpha_i w_i \tag{9}$$

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 almost even 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 is at least $\bar{w}$ ($\leq w_0$). We find $\bar{w}$, observing that end-case blocks have $\alpha = 0$ since they are not partitioned further. Thus from (8) we get

$$\bar{w} = w_n = \frac{\bar{\tau}}{\bar{\tau} + 1} \tag{10}$$

A straightforward way to determine α_i is to assume that they are all equal. This leads to[6]

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

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

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

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

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

and

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

Therefore we can replace the straightforward Equation (11) with

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

[6]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.

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

$$\tau = \frac{1}{\sqrt[n+1]{1 - w_0}} - 1 \tag{15}$$

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.

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

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.[7] While resulting tolerances are likely to increase towards lower levels, they will preserve original lower bounds for relative whitespace in every block[8] 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.

3.3 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

[7] 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.*

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

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 2.[9] Hence, the desired partition capacities C_1 and C_2 need to satisfy $\theta^0 = \frac{C_0 - S_0}{S_0} = \frac{C_1 - S_1}{S_1} = \theta^1$ 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. Formulae (17) are used for computing target partitioning capacities.

4 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.

[9]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.

4.1 Top-down placement testbench

We have implemented a full-fledged global placer that reads standard-cell row-based designs from Cadence Design Systems, Inc. proprietary physical design interchange formats 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 [8] 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' 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.[10] The second method follows Formulae (16), (3), (4) and (5). All overcapacity blocks are treated as explained in Subsection 3.3.

We do not apply "legalization", "cycling", "overlapping" [12] or any other techniques that would move cells between existing blocks or change block boundaries other than during top-down block splits. Without such post-processing the effects of different approaches to whitespace allocation

[10] Which explains our choice of smaller tolerance.

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 with extensive use of the Standard Template Library.[11] Executables are compiled with the `SunPro CC4.2` compiler on Solaris2.6 and optimized with `-O5`. We use Sun Ultra-10 300MHz workstations with 256Mb of memory.

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

Table 1: Statistics for the testcases used in our experiments.

4.2 Results

We report experimental results for six industrial testcases, ranging from $2.7Kcells$ to more than $117Kcells$, with whitespace from 11% to 30% (see Table 1).[12] 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 2.

Our experiments show that

- 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

[11] However, we carefully verified that our code is not slower than equivalent C code when optimized with the `-O5` switch.

[12] White space 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.

	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

Table 2: Placer run-time (CPU seconds on a 300MHz 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.

with significantly 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.

5 Conclusions and future work

We have derived simple formulae for optimal[13] hierarchical whitespace allocation in top-down partitioning-driven placement of standard-cell row-based ASIC 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 in the assumption that all partitioning solutions are legal.

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

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

We suggest that the future work on this subject address the theoretical comparison [16] of balanced recursive bisection with direct multi-way partitioning. In particular, whether direct k-way partitioning for small k allows greater relaxation of partitioning tolerances than recursive bisection and whether the use of direct k-way partitioning can improve final placement wirelength.

References

[1] C. J. Alpert, T. Chan, D. J.-H. Huang, I. Markov and K. Yan, "Quadratic Placement Revisited", *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 752-757.

[2] C. J. Alpert, J.-H. Huang and A. B. Kahng,"Multilevel Circuit Partitioning", *ACM/IEEE Design Automation Conference*, pp. 530-533.

[3] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration*, 19(1995) 1-81.

[4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Relaxed Partitioning Balance Constraints in Top-Down Placement", *In Proc. ASIC '98*

[5] W. Deng, personal communication.

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

[7] S. Dutt and H. Theny, "Partitioning Around Roadblocks: Tackling Constraints With Intermediate Relaxations", *Proc. IEEE International Conference on Computer-Aided Design*, 1997, pp. 350-355.

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

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Design", *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 526-529.

[10] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. Journal* 49 (1970), pp. 291-307.

[11] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE Trans. on Computer Aided Design* 10(3) (1991), pp. 356-365.

[12] , D. J. Huang and A. B. Kahng, 'Partitioning-Based Standard Cell Global Placement with an Exact Objective", *Proc. ACM/IEEE International Symposium on Physical Design*, 1997, pp. 18-25.

[13] David Jensen, Charles Gross, Dinesh Mehta, "New industry document explores defect reduction technology challenges", *Micro Magazine*, January 1998, `http://www.micromagazine.com/archive/98/01/jensen.html`

[14] David Jensen and William Fosnight, "Defect prevention and elimination: Where the rubber hits the road(map)", *Micro Magazine*, October 1998, `http://www.micromagazine.com/archive/98/10/jensen.html`

[15] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*, San Jose, CA, SIA, 1997.

[16] H. D. Simon and S.-H. Teng, "How Good is Recursive Bisection?", *SIAM J. Scientific Computing* 18(5) (1997), pp. 1436-1445.

[17] R. S. Tsay and E. Kuh, "A Unified Approach to Partitioning and Placement", *IEEE Trans. on Circuits and Systems*, 38(5) (1991), pp. 521-633.

[18] Semiconductor Research Corporation, "Physical Design Top Ten Problems", `http://www.src.org/areas/cadts/pd.dgw`