On Whitespace and Stability in Physical Synthesis

Saurabh N. Adya[†], Igor L. Markov[♯], Paul G. Villarrubia[‡]

† Synplicity Inc., 600 W. California Ave., Sunnyvale, CA 95054

♯ The University of Michigan, Department of EECS, Ann Arbor, MI 48109-2122

‡ IBM Corp., 11400 Burnet Road, Austin, TX 78758

Abstract

In the context of physical synthesis, large-scale standard-cell placement algorithms must facilitate incremental changes to layout, both local and global. In particular, flexible gate sizing, net buffering and detail placement require a certain amount of unused space in every region of the die. The need for "local" whitespace is further emphasized by temperature and power-density limits as well as the increasing use of buffered interconnect. Another requirement, the stability of placement results from run to run, is important to the convergence of physical synthesis loops. Indeed, logic re-synthesis targeting local congestion in a given placement or particular critical paths may be irrelevant for another placement produced by the same or a different layout tool.

In this work we offer solutions to the above problems. We show how to tie the results of a placer to a previously existing placement, and yet leave room for optimization. In our experiments this technique produces placements with similar congestion maps. We also show how to trade off wirelength for routability by manipulating whitespace. Empirically, our techniques improve circuit delay of sparse layouts in conjunction with physical synthesis. Our proposed techniques can be implemented using existing commercial placement tools without source code modifications and with modest overhead. They can also be integrated directly into min-cut placers with negligible overhead. We consider in particular detail the problem of scaling existing IP blocks to increase their porosity. Indeed, the need for additional repeater insertion when migrating a block to a newer process node often implies re-optimizing the layout. Our techniques for achieving placement stability allow one to rescale an existing layout with different minimum local whitespace requirements. In contrast to current ECO techniques, our rescaling method is not restricted to small changes of the netlist and layout, but will attempt to keep the relative placements similar if that is possible.

1 Introduction

With the rapid decrease of feature sizes, circuit layouts become more complex, both in terms of size and design constraints [3]. To achieve timing closure for high-performance circuits, it is now common to use physical synthesis — an

approach that combines logic and physical optimization, potentially performing placement-aware buffer insertion, gate sizing, fanout optimization, etc. A recent work from Intel [23] suggests that buffering alone implies the need for "local" whitespace throughout the core area. Such unused cell sites facilitate placement of signal-net and clock-tree buffers in near-optimal locations rather than pre-determined "buffer islands". However, research from IBM [4] shows that distributing whitespace uniformly [8] may significantly increase wirelength. It also suggests that pin-limited and floorplanned designs, e.g., microprocessors with large on-chip caches, very frequently contain placement partitions with large amounts of whitespace. To this end, we (i) develop techniques to achieve a compromise between cell density and design flexibility, and (ii) study relevant trade-offs.

Timing optimization and congestion removal often use loops in which a netlist is re-placed based on information gleaned from a trial placement. However, some popular algorithms such as min-cut placement and simulated annealing tend to produce very different placement solutions from run to run. Therefore information about timing-critical nets and nets that failed to route may be invalidated (similar reasons hamper interconnect prediction [25]). To facilitate incremental improvement of layout, we propose to stabilize placements from run to run. We distinguish two kinds of stability. An *inherently stable* algorithm, such as many analytical algorithms, would produce similar results from run to run. However, even with a generally unstable algorithm one can *tether* all new placements to a given trial placement, with a tunable amount of freedom for further optimizations. Thus, we distinguish *inherent stability* from *relative stability*. The latter may be used, e.g., to tie placements produced by an annealer to a placement produced by a min-cut algorithm. We demonstrate such relative stability by comparing congestion maps [19] of several min-cut placements, cell displacements and top critical paths. Our empirical results show that small modifications of a placement instance can suppress the instability inherent in common placement algorithms, without the loss of solution quality. Our techniques rely largely on pre- and post-processing, and can be easily implemented with existing tools.

Another trend in VLSI design is the increasing dominance of interconnects [13]. This is primarily because the wires do not scale as well as the devices. Assuming ideal scaling, all dimensions of a wire are shrunk by $0.7x$ per generation.[1] It is known that the wire capacitance per unit length remains invariant from generation to generation [5]. However, the resistance per unit length doubles every process generation, resulting in a wire delay which scales as $1.4x$ every generation. RC delay of a wire grows quadratically with the length of the wire. Repeaters are often placed at optimal distances (generally equal) on the wire to linearize the delay through the wire [5]. Since for an ideally shrunk interconnect, the wire delay scales as $1.4x$, it implies that more number of buffers are required for an interconnect to linearize the delay through it in the new process generation. Additionally, the design frequencies are also increasing causing the number of buffers to increase per process node. The requirement of additional number of buffers would often entail replacing the entire block with a more relaxed minimum local whitespace requirement. In an application of our proposed techniques on placement stability, we address the issue of rescaling a placement to satisfy different minimum local whitespace requirements while maintaining the timing characteristics of the original placement. Our rescaling flow is not restricted to small changes of the netlist and layout unlike current ECO techniques. By combining our techniques one can devise

[1] Currently the wire heights do not scale as well as the width, resulting in tall thin wires

placement flows to efficiently map an existing layout-optimized design to a new process generation, while allowing sufficient room for further optimizations. Companies selling soft IP, like Tensilica and MIPS, and their customers could benefit from rescaling. Microprocessors are also often downscaled. Perhaps our techniques can be used as a first step followed by technology-specific and/or performance driven layout and circuit optimizations.

In the remainder of the paper, Section 2 gives background on large-scale placement and describes previous work. Whitespace distribution is discussed in Section 3, and stability in Section 4, accompanied by relevant empirical results. Our proposed rescaling flow to allow for different minimum local whitespace requirements is explained in Section 5. Finally, our contributions are summarized in Section 6.

2 Background and Previous Work

Modern ASIC designs are typically laid out in the *fixed-die context*, where the outline of the core area, all routing tracks and power lines are fixed before placement starts [6]. One of the reasons for this is the use of previously designed and rigorously simulated power grids. Also, standard-cell partitions of microprocessors are often laid out with fixed outlines in hierarchical floorplan-driven design flows because reshaping the outline would affect neighboring partitions. Large on-chip memories similarly constrain random-logic partitions. In the context of massive IP reuse, especially with hard IP blocks (analog circuits such as DACs, ADCs, PLLs and embedded memories), the die area may be determined by floorplanning, thus making area-minimization during placement irrelevant. Fixed-die layout is reasonable for processes with over-the-cell routing on three or more metal layers. In this context, the total area is fixed and the number of unused cell sites — whitespace — is known in advance. Variable-die placers typically pack all cells to the left in rows. However, fixed-die placers often allocate whitespace uniformly [6, 8] or according to congestion maps [21, 27]. When significantly more whitespace is available, the work in [4] proposes to allocate whitespace so as to improve half-perimeter wirelength. They show that uniform whitespace distribution in such designs causes very significant increase in wirelength.

2.1 Fixed-die placement in physical synthesis

It is important to note that in the context of physical synthesis, the structure of the netlist may be changed and incremental placement must be performed. Given that some gates may be up-sized and many nets are likely to be buffered, the availability of “local” whitespace is a necessity. Indeed, the work in [23, 17] predicts that buffers will soon be the most frequently used gates in large high-performance circuits. Local whitespace can also be useful to accommodate regular structures such as (i) N-well contacts that have to be assigned to vertices of a grid, and (ii) area-array I/O pads that also form a grid. Thus, desired whitespace distribution must guarantee a minimum percent of “local” whitespace throughout the chip and beyond that optimize other design objectives. The requirement for minimum local whitespace may also be used to generically improve routability and yield, even out the temperature gradient across the die and decrease the likelihood of cross-talk noise.

Another effect of fixed-die layout is the occurrence of unroutable placements. Indeed, in variable-die layout one can always add routing tracks to complete routing at the cost of increased area [20], but this is impossible with a fixed outline. To improve congestion, it is common to use cell-bloating (i.e., treating cells as if they were larger in order to free routing tracks around them) in congested regions [24]. Additionally, a number of logic transformations (fanout optimization, input reordering, gate merging and cloning, etc) can be used to improve congestion. However, if the same placement tool produces an entirely different placement at the next run, such optimizations would be wasted. This problem is especially noticeable with placers based on min-cut and simulated annealing. The same problem is encountered when logic re-synthesis targets timing optimization. Therefore, to reliably achieve timing closure one may want to stabilize placement solutions.

2.2 Hierarchical Whitespace Allocation in Top-Down Placement

The academic placement tool Capo [6] applies a top-down, min-cut partitioning based approach to find a global placement. Capo uniformly spreads [3] the available whitespace throughout the core region. We briefly explain the whitespace allocation strategy implemented in Capo [8]. In the top-down, divide-and-conquer approach for global placement, a given placement instance is decomposed into smaller instances by subdividing the placement region and assigning cells to sub-regions such that good solutions to sub-instances combine into good solutions of the original instance. The concept of a *placement bin* is pivotal. A bin 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, and their locations are fixed. In a min-cut placer like Capo, every placement bin yields a hypergraph partitioning instance which is split through min-cut hypergraph bisection with FM-type move-based heuristics. The uniform whitespace distribution strategy in Capo is explained as follows. Let a placement bin have *site area S*, *cell area C*, *absolute whitespace* $W = max\{S-C,0\}$, and *relative whitespace* $w = W/S$. A hypergraph bi-partitioning solution implies cell areas C_0 and C_1 in child bins, such that $C_0+C_1 = C$, $0 \le C_0, 0 \le C_1$. The input to the hypergraph bi-partitioner must specify both the netlist and the allowed ranges for C_0 and C_1, i.e., bounds $C_0^{min} \le C_0 \le C_0^{max}, C_1^{min} \le C_1 \le C_1^{max}$. These bounds establish *absolute* tolerance $T_j = C_j^{max} - C_j^{min}$ and *relative* tolerance $\tau_j = T_j/C$. Capo uses a mix of fixed tolerances and hierarchical whitespace allocation during top-down placement [8]. The placer chooses vertical or horizontal bin splits depending on the bin's aspect ratio and typically cuts along the longest side of a bin. Vertical partitioning is performed with a fixed 20% tolerance. After partitioning, when the actual total cell area in each partition is available, the vertical cut-line determining the bin boundaries is shifted to equalize relative whitespace in the bins. A different strategy is employed for allocating whitespace for bins split by horizontal cut-line. During a horizontal split, the partitioning tolerances are calculated based on the relative whitespace of the bin and the number of rows in the bin. A precise mathematical model of hierarchical whitespace allocation in placement is proposed in [8]. It is based on the concept of *whitespace deterioration* which is explained as follows. Assuming non-zero relative whitespace at top-level, one will require that for each bin split with

a relative whitespace of w, the relative whitespace in each child bin is at least αw, where $0 \leq \alpha \leq 1$ is the *whitespace deterioration*. As α approaches 1, the whitespace distribution in the final placement approaches uniform distribution. An α of 0 allows for fully utilized regions of the layout. One can adjust α on a per-bin basis to account for maximum allowed layout densities in the leaf-level bins which can be guided by minimum local whitespace requirements. It is shown in [8], that given the whitespace deterioration α for a bin, the partition capacities and tolerances for the partitioner can be calculated as follows.

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

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

$$C_0 \geq max\{0, C - C_0^{max}\} =: C_0^{min}$$

$$C_1 \geq max\{0, C - C_1^{max}\} =: C_1^{min}$$

When the bin has large amount of whitespace (i.e. C is very small compared to S) and α sufficiently small, C_j^{max} and C_j^{min} may degenerate into C and zero, respectively. In such a case, all cells are allowed to go into one partition. A closed expression for whitespace deterioration α in terms of relative whitespace w in the bin and the number of rows R in the bin is given as follows.

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

Partitioning tolerances increase as the placer descends to lower levels, and relative whitespace in all bins is limited from below, thus preventing overlaps. This facilitates good use of whitespace, when it is scarce and prevents dense regions when large amounts of whitespace are available. Similar to vertical partitioning, after horizontal partitioning, the cut-line determining the bin boundaries is shifted to equalize relative whitespace in the bins. Hierarchical whitespace allocation during horizontal partitioning allows for higher tolerances during partitioning, thus allowing for lower cut [15] during min-cut operation. This can also lead to certain regions of the layout being packed more densely than others. However, a constant tolerance during the vertical partitioning step and shifting of cut-lines after each partitioning to equalize the relative whitespace in the child bins ensure a uniform distribution of whitespace through-out the core region.

3 Whitespace Management Framework

As shown in [4], min-cut placers that uniformly distribute whitespace [8, 6] tend to produce excessive wirelength when large amounts of whitespace are present. The authors of [4] propose a fairly sophisticated technique, Analytical Constraint Generation (ACG), to place sparse designs. It has been argued in [4] that analytical placement algorithms have a global view of the placement problem and can better manage large amounts of whitespace. ACG [4] combines a min-cut based placer with quadratic placement engine. In ACG, during top-down recursive bisection based min-cut placement flow, the partitioning capacities for each placement bin to be partitioned are generated based on quadratic wirelength minimum placement at that level. While we address the same problem of placing sparse designs, our study is somewhat orthogonal

to theirs. The methods we propose are much simpler and can be implemented as pre-processing without having access to placer source code. This allows us to explore the effect of whitespace on routed wirelength and congestion using different academic placers. We also describe optimized implementations of our whitespace management techniques in a typical top-down min-cut placer framework. Additionally, our placement framework is somewhat different from that used in [4] and benefits from these simple techniques in new ways. Namely, Capo can shift the cut-line to better reflect the outcome (balance) of every min-cut partitioning call, whereas the placer in [4] uses a grid of placement bins rather than a more general slicing floorplan as in Capo.

3.1 Free Cells

The technique we propose assumes a placer that uniformly distributes whitespace across the core area. We assume that the minimum "local" whitespace requirement leaves certain slack relative to the total whitespace available in the design. By pre-processing, we can ensure (i) the minimum "local" whitespace through the core area, and (ii) better allocation of the remaining whitespace. The technique consists of adding small disconnected "free cells" to the design in an amount not exceeding whitespace that remains after the "local" requirement is satisfied. Since free cells are disconnected and small, a placer is free to place those cells so as to improve relevant design objectives. After placement, we remove free cells and treat the remaining cell sites as empty. This causes high cell density in certain areas, with free cells occupying the vacant areas of the chip.

Our empirical evaluation uses the Capo placer [6] which uniformly distributes available whitespace [8] with routability in mind (`Feng Shui 2.0` and `mPl 2.0` do not distribute whitespace, but `Dragon 2.23` does in the fixed-die mode). However, with designs having low placement densities this strategy results in excessive wirelength and potentially poor signal delay. Figure 1 shows placements of an industrial design with 72940 cells, 73155 nets and 74% whitespace. Figure 1 (A) shows the placement achieved by uniform distribution of whitespace and Figure (B) shows the placement achieved by introducing free cells to reduce whitespace available to the placer to 15%. The wirelength of the design was improved from 15.32e6 to 8.77e6. The global placement runtime increased from 444 seconds to 722 seconds. ACG was also tested on this circuit [4], and wirelength improved from 11.43e6 (for uniform whitespace distribution) to 10.38e6 (with ACG). Figure 2 shows the effect of free cells on the local whitespace distribution for the same design. To calculate the local whitespace distribution, we divide the layout region into a grid of bins (27x27 in this case) and calculate the local whitespace in each bin. Free cells are removed from the design before calculating the local whitespace distribution. We plot the % of bins vs. the % local whitespace in each bin. As seen from Figure 2, with no free cells introduced during the placement, most of the bins have a local whitespace of around 70-80%. When free cells are added to the design to reduce whitespace to 30%, a large number of bins have 100% whitespace. These bins represent the vacant areas of the chip as seen in Figure 1 (B). However, most of the bins containing standard cells have a local whitespace around 30%. Similar effect is observed when free cells are added to reduce whitespace to 15%. While we have not performed experiments with ACG, we suspect that ACG may further improve wirelength if used in conjunction with free cells. This

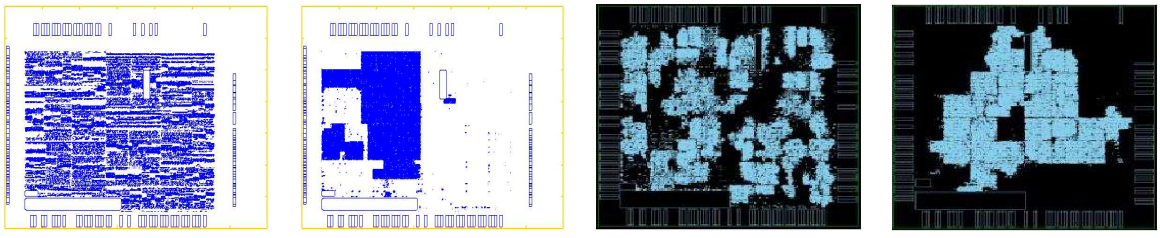
can be demonstrated using a sparse design with one dense cluster of logic connected to pins on the periphery so that the cluster must be placed in the center to minimize wirelength. However, since such a placement implies a high top-level cut, some top-down placers (especially those with fixed cutline) will avoid this optimal placement.

We show that better whitespace allocation reduces wirelength in mixed-size placement flow from [2]. The main contribution of [2] is a methodology to place designs with numerous macros by combining floorplanning and standard-cell techniques. The proposed design flow is as follows:

- A black-box standard-cell placer generates an initial placement. In a pre-processing step, all macros are shredded into small pieces (fake cells) connected by fake wires, and pins from the macro are propagated to individual pieces. Each macro is thus represented by a grid, and the resulting netlist consists of only small cells. If the fake nets have sufficiently high weights, the fake cells belonging to the same macro should place next to each other. Fixed orientations of macros can be accommodated.
- The initial locations of macros are produced by averaging the locations of respective fake cells. To remove overlaps between macros, a physical clustering algorithm constructs a fixed-outline floorplanning instance. Thus, small standard cells placed next to each other are clustered and form soft blocks.
- A fixed-outline floorplanner [1] generates valid locations of macros and soft blocks of movable cells.
- With macros considered fixed, the black-box standard-cell placer is called again to re-place small cells.

Step 4 of the mixed-size placement flow presented in [2] fixes the macro locations to the ones provided by the floorplanner and replaces standard-cells around the macros. We improve whitespace allocation in this stage by introducing free cells. We add free cells to reduce the available whitespace to the placer to 10% and replace the design with the macros being fixed. The results are summarized in Table 1. We compare our results to mPG [11].

Physical synthesis flows interleave placement optimizations with logic optimizations to achieve desired timing. This reduces the number of iterations required between the front-end design and back-end design for timing closure. Physical synthesis tools typically start from a global placement and perform logic optimizations like buffer insertion, driver sizing,



(A) (B) (C) (D)

Figure 1: **The `ckt4` design from IBM has 72940 cells, 73155 nets, several pre-placed macros and 74% whitespace. Figure (A) shows a placement produced by Capo with uniform whitespace distribution. Figure (B) shows another placement produced by Capo after free cells were added to reduce placer whitespace from 74% to 15%. This reduces the half-perimeter wirelength from** $15.32e6$ **to** $8.77e6$**. Free cells are not shown in the placed design. Figure (C) shows a placement obtained from a min-cut placer from IBM and (D) shows the placement obtained by the ACG technique [4].**

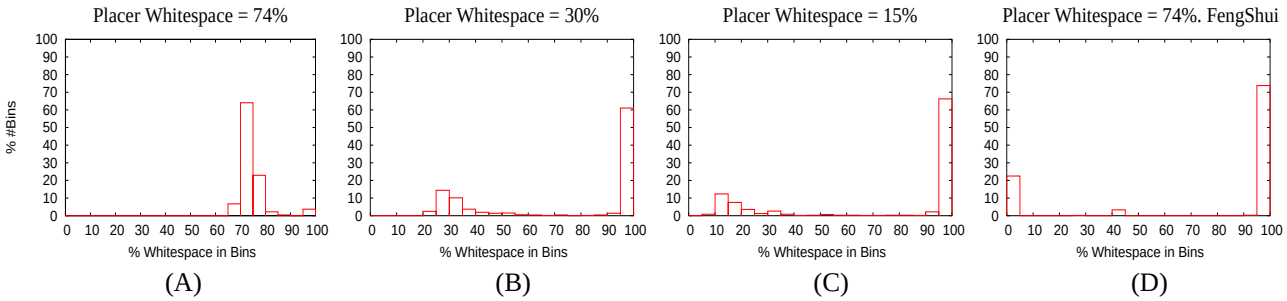


Figure 2: **A histogram for local whitespace in design ckt4 from IBM with 74% whitespace. We subdivide the core area into a 27x27 grid and calculate local whitespace in each bin. We plot the % number of bins versus the % whitespace in each bin. Figure (A) shows the local whitespace distribution in a Capo placement with no free cells added. Figure (D) shows a similar distribution for Feng Shui 2.0 that has no whitespace management and packs cells to the left. Figures (B) and (C) show the local whitespace distribution achieved by Capo with free cells added to reduce the whitespace available to the placer. Free cells are removed after placement for local whitespace computation.**

logic replication etc. to improve timing of the design. These logic optimizations are based on the physical information generated by the initial global placement. Such tools rely on ECO placement techniques to legalize incremental changes in the netlist after global placement. This enforces a minimum local whitespace requirement after global placement to facilitate ECO placement after changes due to logic optimizations. Compacting a placement without physical synthesis in mind will severely limit the efficacy of the physical synthesis tools. We study the effect of free cells on physical synthesis in Table 2. We conduct our experiments on proprietary industrial benchmarks with varying row-utilization. We report the worst slack and the total negative slack (TNS) in the design after the physical synthesis. In the default run, the global placer (Capo) uniformly spreads the cells around the core area. As an alternative flow, we add free cells during the global placement stage to reduce the whitespace available to the placer to 40%. Thus, the global placer compacts the placement but ensures minimum local whitespace of 40% around the core area. Free cells are removed after global placement. As seen from the results in Table 2, the worst slack and total negative slack for all the designs improve considerably by adding free cells during the global placement stage of physical synthesis. All the designs are routable even after compacting the designs by using free cells.

We also conduct experiments to demonstrate the effect of free cells on the routability of a design. We use the ibm02 benchmark from [27]. The design initially has about 9% whitespace. The design is re-floorplanned to have 65% whitespace. The design is placed with Capo placer and routed with WarpRoute from Cadence. Free cells are gradually added during placement, reducing whitespace that the placer can allocate uniformly. Each of these designs is placed; the free cells are removed after placement and the design is routed with Cadence WarpRoute. Table 3 reports the results of these experiments. Clearly, adding free cells consistently improves half perimeter wirelength. The routed wirelength and routing time also improve initially because of better placed wirelength. However, after a certain threshold, routed wirelength increases and then the designs become consistently unroutable. Thus, free cells are useful in reducing the half-perimeter wirelength, but, distributing a portion of whitespace uniformly helps Capo produce routable placements. In fact, report-

Circuit	Flow = Capo+Parquet+Capo [2] (High Temp Anneal) I Uniform WS			Our Flow = Capo+Parquet+Capo (Low Temp Anneal) II Uniform WS			III Uniform WS + Free Cells			mPG[11] IV	
	HPWL(e6)	Time	#FP Tries	HPWL(e6)	Time	#FP Tries	HPWL(e6)	Time	#FP Tries	HPWL(e6)	Time
ibm01	3.96	18m	1	3.36	13m	1	3.05	20m	4	3.01	18m
ibm02	8.37	31m	1	8.23	4hr0m	15	6.83	11m	1	7.42	32m
ibm03	12.16	42m	1	11.53	22m	1	10.38	59m	6	11.2	32m
ibm04	13.48	47m	1	11.93	25m	1	10.11	15m	1	10.5	42m
ibm05	11.51	8m	N/A	11.20	5m	N/A	11.1	5m	N/A	10.9	36m
ibm06	10.25	56m	3	9.63	19m	1	9.94	18m	1	9.2	45m
ibm07	15.75	58m	1	15.80	39m	1	15.25	25m	1	13.7	1hr8m
ibm08	21.18	1hr34m	1	18.85	1hr51m	3	17.91	29m	1	16.4	1hr22m
ibm09	19.59	1hr6m	1	17.52	2hr58m	6	19.88	29m	1	18.6	1hr24m
ibm10	60.72	3hr49m	1	53.58	8hr10m	3	45.46	1hr56m	1	43.6	2hr52m
ibm11	28.49	1hr46m	1	26.47	1hr9m	1	29.4	45m	1	26.5	1hr52m
ibm12	51.74	11hr15m	4	55.12	1hr59m	1	55.79	25m	1	44.3	1hr33m
ibm13	39.39	2hr31m	1	33.56	1hr28m	1	37.73	53m	1	37.7	1hr31m
ibm14	56.19	4hr46m	1	52.67	5hr33m	2	50.26	2hr35m	1	43.5	4hr36m
ibm15	70.48	3hr57m	1	64.69	4hr24m	2	65.0	3hr15m	1	65.5	6hr25m
ibm16	-	-	-	83.14	9hr40m	4	90.01	2hr42m	2	72.4	7hr16m
ibm17	92.38	7hr23m	1	91.50	4hr9m	1	89.17	3hr8m	1	78.5	10hr6m
ibm18	54.90	5hr78m	2	54.11	6hr37m	5	51.84	2hr7m	1	50.7	7hr17m

Table 1: **Mixed-size placement (Capo+Parquet+Capo), with the floorplanner Parquet using low-temperature annealing to preserve initial macro locations. We report results for uniform whitespace distribution without free cells (II) and with free cells (III). Results are compared with the high-temperature annealing flow from [2] with uniform whitespace distribution (I) and mPG (IV). Runtimes for Table I are observed on 1 GHz Linux/Pentium 3 machine and are reproduced from [2]. Runtimes for Table II and III are observed on a 2 GHz Linux/Pentium 4 machine. Runtimes for mPG (IV) are observed on a Sun Blade 1000 workstation running at 750 MHz and are reproduced from [10].**

Circuit	#Cells (During Placement Stage)	no Free Cells						w Free Cells					
		After Placement				After Phy-Synthesis		After Placement				After Phy-Synthesis	
		Place %WS	Place RunTime (sec)	WL	Worst Slack (ns)	Worst Slack (ns)	TNS (ns)	Place %WS	Place RunTime (sec)	WL	Worst Slack (ns)	Worst Slack (ns)	TNS (ns)
Ind1	10957	89	38	4.29e6	-3.75	-0.116	-2.150	40	75	3.37e6	-2.62	0.046	0.00
Ind2	39600	60	185	4.67e6	-7.70	-2.14	-6975	40	223	4.08e6	-6.57	-0.951	-2854
Ind3	109558	81	818	2.71e7	-14.77	-8.67	-133467	40	1562	1.51e7	-9.89	-2.19	-47578

Table 2: **The impact of free cells on physical synthesis for industrial designs with low utilization. We report the worst slack and total negative slack (TNS) after physical synthesis. During the placement stage of physical synthesis, we add free cells so that the whitespace available to the placer was reduced to 40%. Free cells are removed after global placement. All designs are routable after physical synthesis.**

ing only half-perimeter wirelength may be misleading. Routability of Capo and Dragon placements on `ibm-Dragon` benchmarks is discussed in [3], where, the differences are traced to greater horizontal wirelength and smaller vertical wirelength in Capo placements.

3.2 Low-Overhead Implementation of Free Cells in a Min-cut Placer

As explained in Section 3, a generic whitespace management framework can be obtained by using a placer that distributes whitespace uniformly through the core region and by representing the excessive whitespace as small disconnected free cells. This implementation of free cells does not require any changes to the placer source code and only pre-processes the input netlist by introducing fake free cells, which makes sense with existing commercial placers. However, explicit modeling of free cells and letting the placer process the modified netlist impacts the run-time and the memory footprint of the placer. The placer run-time degradation is evident from results presented in Tables 2 and 3. In this section, we

%Free Cells	Capo 8.6 Place WL(e8)	Place Time(s)	Routed WL(e8)	Route Time(s)	#Vio lations	Route Success	Dragon 2.23 (fi xed-die mode: -fd) Place WL(e8)	Place Time(s)	Routed WL(e8)	Route Time(s)	#Vio lations	Route Success
0	1.80	129	2.29	2160	1	Yes	1.97	1618	2.42	1020	0	Yes
5	1.68	130	2.14	1080	0	Yes	1.89	1611	2.37	780	0	Yes
10	1.68	152	2.22	1800	0	Yes	1.83	1348	2.31	1560	1	Yes
15	1.64	162	2.77	1680	20741	No	1.67	1921	2.07	600	0	Yes
20	1.57	168	2.90	2040	27883	No	1.66	2342	2.17	720	0	Yes
25	1.55	186	2.95	2640	63864	No	1.57	2030	2.09	780	0	Yes
30	1.52	181	3.00	1560	66096	No	1.52	1988	2.12	1380	0	Yes

Table 3: **Place and Route results for `ibm02` benchmark from IBM-Dragon suite with whitespace increased to 65%. Free cells (as a fraction of total area) are added to handle whitespace. Capo allocates the remaining whitespace uniformly. Dragon performs congestion-driven allocation of the remaining whitespace. All experiments are conducted on a 2GHz Pentium/Linux platform.**

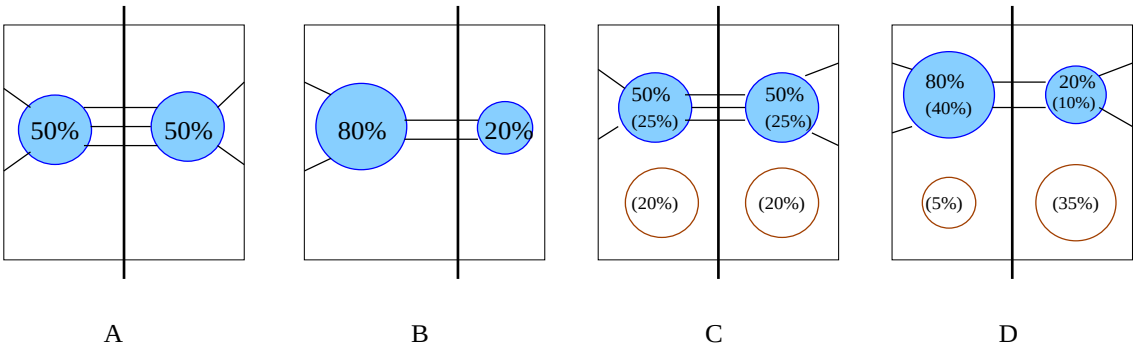
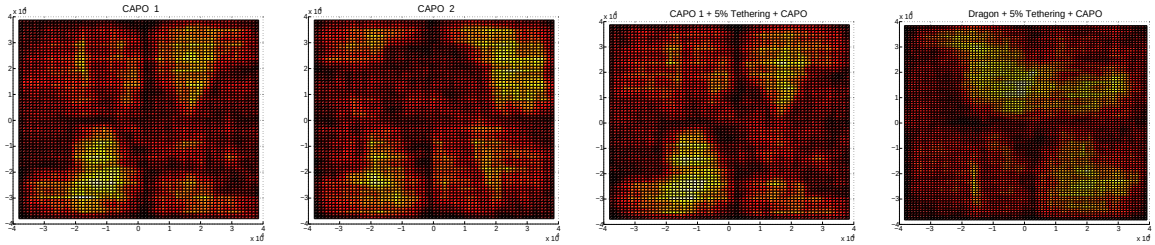


Figure 3: **A bin with 50% whitespace being partitioned. Figures A and B have only real std-cells. Figures C and D have free cells introduced to occupy 40% of site area leaving 10% whitespace to be allocated by the partitioner. Real std-cells are shown in shaded circles and free cells are shown in plain circles. Numbers in circles without parentheses are area of cells as % of total real std-cell area in the bin. Numbers in parentheses are area of cells as % of total site area in the bin.**

explain implicit handling of large amounts of whitespace in a top-down, recursive bisection based min-cut placer [6] with minimal runtime and memory overhead. We achieve this without representing excessive whitespace as free cells and hence without pre-processing the input netlist.

Figure 3 shows the min-cut partitioning procedure for a bin with 50% whitespace in absence and presence of free cells. Figures 3A and B show 2 ways to partition a bin with no free cells added. The parameters affecting the balance in the two partitions are (i) Partition capacities (C_j), and (ii) Partition tolerances (C_j^{max},C_j^{min}). In the example, Figure 3A shows the std-cells being partitioned in the ratio 50%:50% with a net-cut of 3. However, if higher tolerance is allowed the partitioner may decide to partition the cells in the ratio 80%:20% with a smaller net-cut of 2. As shown in Figure 3B, Capo is allowed to shift cut-lines after partitioning to equalize the relative whitespace in the two partitions. Figures 3C and D show how the partitioning procedure works in presence of free cells. Out of the 50% whitespace, 40% is represented as free cells and the remaining 10% whitespace can be distributed between the two bins. To achieve a lower cut, a good min-cut partitioner will favor to partition the instance as shown in Figure 3D over Figure 3C. Thus, the effect of higher tolerances is imitated using free cells and a constant tolerance.

As explained in Section 2.2, Capo uniformly spreads the available whitespace throughout the core region. Capo uses



(A) (B) (C) (D)

Figure 4: **Placements of the `ibm02` design with 19321 cells and 18429 nets, 9% whitespace and no terminal connections. Figures (A) and (B) show congestion maps of `ibm02` placed by two different runs of Capo. As seen, the congestion maps are different indicating the lack of stability in the placement algorithm. Figure (C) shows the congestion map of a placement produced by tethering 5% of movable cells to the seed placement in Figure (A) and running Capo again. Figure (D) shows the congestion map of a placement produced by tethering cells to a placement produced by Dragon and then running Capo on the tethered netlist.**

hierarchical whitespace tolerance calculation [8] only while splitting a bin horizontally. The tolerance during the vertical split is constant and the vertical cut-line is allowed to move after partitioning to balance the relative whitespace in the two child bins. This strategy works well for low whitespace designs. However, for high whitespace designs, it results in lower tolerances during vertical partitioning and uniformly distributes the whitespace in the core region resulting in excessive wirelength. After studying the behavior of Capo8.7 on low utilization designs, we added the option -nonUniformWS to Capo8.8. This causes Capo to use the same hierarchical tolerance computation for both horizontal and vertical splits when the bin whitespace is greater than the minimum local whitespace requirement. Since, during the top-down placement process, the aspect ratio of most of the bins is close to 1.0, we can approximate the number of recursively applied parallel vertical bin splits to $n = log_2 R$, where R is the number of rows in the bin. With this assumption, the partition tolerances are calculated in the same manner for horizontal and vertical splits. This change allows Capo to transparently handle designs with a large amount of whitespace. To account for local minimum whitespace requirement, we make sure that C_j^{max} for any partition does not violate the local minimum whitespace requirement for that child bin. Also, if the bin whitespace is greater than the minimum local whitespace requirement, we do not shift the cut-lines after partitioning. This ensures that some regions of the layout are more tightly packed than other regions resulting in lower wirelength. However, the minimum local whitespace requirement are still respected. We test the effect of this change on the "qor" test case from IBM which has 73095 cells ,73155 nets and 74 % whitespace in the design. Capo8.7 distributes the whitespace uniformly around the chip and produces a placement with Half Perimeter Wirelength (HPWL) of 15.85e6. With our changes, Capo8.8 allows higher tolerance during the initial cuts, having the effect of compacting the placement. The final HPWL of the placement produced by Capo8.8 with a 15% minimum local whitespace requirement is 8.9e6. The performance of Capo8.8 on the IBMv1 and IBMv2 benchmarks remains unchanged with respect to Capo8.7. This is to be expected as these benchmarks have artificially-created layout regions with a relatively small amount of whitespace.

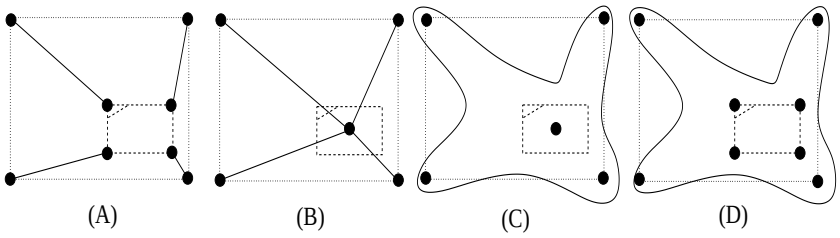


Figure 5: **A single cell/macro is tied to a rectangular region in 4 different ways. Solid dots show artificially added (fake) pins, skew lines show fake two-pin nets, and a fake 5-pin net is shown by a spline. In all three cases moving the cell within the region does not affect the total length of fake nets. However, any placement beyond the region will incur a wirelength penalty that is independent of other movable objects.**

4 Stability of Placement Results

Physical synthesis flows often require the stability of placement results from run to run for future optimizations which target timing and/or congestion. However, Figures 4 (A) and (B) show that congestion maps [19] produced for unrelated runs of a randomized min-cut placer may be very different. In order to improve congestion, one may distribute whitespace to congested areas or restructure the logic, but such fixes may be irrelevant to the result of the next placement run, or if another placer is used. To achieve relative stability, we propose the following approach. Given a placement, we modify the original netlist by adding fake pins and fake nets. After the modified netlist is placed, the locations of real cells are likely to be close to their original locations, and the amount of change allowed can be easily controlled during pre-processing. It is important to note that we are not adding hard constraints — in principle, any cell can be placed anywhere. However, locations that are far from the original location carry a wirelength penalty in terms of fake wires — further the location, greater the penalty. A key property of our construction is that all locations within a prescribed rectangle centered around the original location carry the same minimal wirelength penalty, and this are equally attractive during wirelength optimization.

Figure 5 demonstrates several ways to tie a cell or a macro to a region without inducing a hard constraint. Four outer fake pins are fixed in the corners of the given region. In Figure 5 (A), four fake pins are added in the corners of the cell to preserve cell orientation. In Figure 5 (B), one fake pin is added at the center of the cell so that changes in orientation do not affect wirelength. In Figure 5 (C), the same effect is achieved by using one fake 5-pin net rather than four fake two-pin nets. In Figures 5 (B) and 5 (C), only the center of the cell is constrained to be in the region. In Figure 5 (D), one fake 8-pin net is used with the fake pins in the corners to ensure that the entire cell is placed within the region. Note that a technique similar to that in Figure 5 (A) is used in [2] to restrict orientations of macros, but in that work the four outer fake pins are fixed at the corners of the core region. The three new constructions ignore cell orientations. The first one uses four two-pin nets, the second uses one five-pin net and the third uses one eight pin net. The third new construction was suggested to us by Amir Farrahi from Sun Microsystems. It can be used to mitigate the number of added nets and to ensure that the entire cell is placed within the constraining region. Otherwise, these constructions are equivalent if used with min-cut placers or placers based on simulated annealing that minimize HPWL.

% Rgn Size (of layout)	% Avg Diff	%Max Diff
0.2	3.3	47
0.5	2.6	58
1.0	3.9	50
10.0	3.7	43
50.0	5.1	48

Table 4: **The impact of constraining region size during tethering on the stability of global placements produced by Capo on the ibm06 benchmark. The constraining region size is measured as a percent of the total layout region size. 5% of cells are tethered to the base placement for all the runs. We report the average and maximum Manhattan displacement per cell between tethered placements and the base placement.**

In our experiments, we randomly select 2%-5% cells in a given placement and tie them to regions centered at the cells' locations. The size of the regions is selected as a small fraction (several percent) of the core region size. These sizes and the weights of fake wires allow one to control changes from the original placement. As shown in Figure 4 (C), additional runs of the min-cut placer Capo produce essentially the same congestion map. The placement in Figure 4 (D) is tied to the output of Dragon. Table 5 reports the effect of tethering cells to a base placement on the IBM-v1 benchmarks [26]. Base placements are generated using the randomized min-cut placer Capo. We then tether a small number of randomly selected cells of the netlist to the base placement. The IBM-v1 benchmarks have disconnected groups of cells, caused by the removal of macros (and incident nets) during the conversion from the original ISPD 98 partitioning suite to placement benchmarks [26].[2] To stabilize such designs we randomly select at least one cell from each disconnected component in the netlist for tethering. Table 5 reports the average and maximum Manhattan difference between locations of cells in the new tethered placements to those in the base placement. The difference is reported as a percentage of the core region bounding box and can be compared to the tethering region whose half-perimeter is 1% of that bounding box. As seen from the results, tethering several % of the cells to a base placement dramatically improves the stability of the randomized min-cut placer — the average cell displacement from the initial locations is very small. However, the maximum displacement remains comparatively high. We trace this to cells in high fanout nets which, if not tethered, have a large freedom to be placed around the core region without affecting the half-perimeter wirelength of the design.[3] In practice, when it is desirable to stabilize placement with respect to a particular design objective, e.g., circuit delay, one should tether cells that are relevant to that objective, e.g., those on critical paths (see Section 5.2).

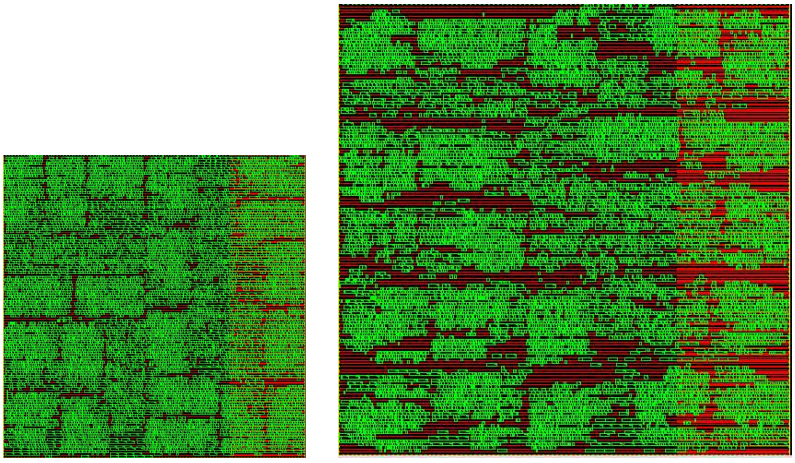
Table 4 shows that the constraining-region size does not have a significant effect on the stability of global placement as measured by average and maximum displacement — a surprising result. Finally, in all of our experiments, except for those with very small constraining regions, the wirelength of tethered placements is similar to the original wirelength.

[2] Similar disconnected cells and groups of cells also occur in some real-world design methodologies, e.g., "bonus cells" that are sprinkled through designs in anticipation of future incremental changes.

[3] We attempted adding cells with largest displacements to the list of tethered cells and re-running the placer. On our benchmarks this approach has only moderate effect because it takes a number of iterations to identify all "loose" cells.

Circuit	#Cells	#Nets	No Tethering		2% Cells Tethered		5% Cells Tethered		10% Cells Tethered		50% Cells Tethered	
			%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff
ibm01	12282	11507	46	98	6	38	2.6	32	3.1	38	1.1	39
ibm02	19321	18429	24	58	4.8	47	4	45	2.9	54	1.1	37
ibm03	22207	21621	19	92	6.1	61	3.1	44	2.6	59	1.6	41
ibm04	26633	26163	41	95	9.4	58	3.3	45	2.9	51	1.3	51
ibm05	29347	28446	7.6	70	6.4	86	4.2	87	3.1	68	1.5	82
ibm06	32185	33354	40	95	5.3	53	3.8	62	2.7	48	1.4	44
ibm07	45135	44394	14	90	4	43	3.1	42	2.1	55	1.4	41
ibm08	50977	47944	36	90	2.7	56	2.1	59	1.7	56	0.9	59
ibm09	51746	50393	38	89	5.6	45	2.7	44	1.9	38	1	26
ibm10	67692	64227	23	94	3.2	56	1.8	50	1.4	50	0.6	49

Table 5: **The impact of tethering on stability of global placements produced by the Capo placer. Using `ibm-v1` benchmarks, we evaluate the impact of tethering random 2% / 5% / 10% / 50% of cells to a base placement. We report the average and maximum Manhattan cell-to-cell displacement between tethered placements and the base placement. The displacement is reported as % of the core bounding box.**



(A) Row Util=70% Aspect Ratio=1

(B) Row Util=50% Aspect Ratio=1

Figure 6: **Capo placements for AES(Rijndael) core.**

5 Application: Resizing Existing Placed Designs

The number of buffers will, in general, increase as we move to lower process nodes [23]. This is primarily because (i) wires are not scaling as well as devices, (ii) transistor counts on chips are increasing, resulting in more number of buffers per logic gate. As we scale to newer process nodes and existing IP blocks gets embedded in larger designs, the IP blocks may have to be more porous, i.e. have a larger minimum local whitespace requirements. We extend our techniques for achieving placement stability to the context of rescaling a layout while maintaining the same relative timing characteristics. In contrast to current ECO techniques, our proposed rescaling method allows for large-scale optimizations during the placement process and is not limited to small changes in netlist and layout.

Figure 6 shows 2 different block shapes (floorplans) for the same circuit AES(see Section 5.3). Figure (A) shows the design with 70% row utilization and an aspect ratio of 1. Figure (B) shows the design with 50% row utilization and an

aspect ratio of 1. Our objective in this work is the following : given a layout optimized design (Figure 6 (A)); rescale this design to fit a larger die (Figure 6 (B)) resulting from higher local minimum whitespace requirements.

We use the techniques for achieving stability presented in Section 4 as the basis of our rescaling technique. We tether randomly chosen x% of standard cells to the locations specified by the original layout. This helps a decision based placer such as top-down min-cut placer to make the right decisions while splitting the netlist at each partitioning level. Thus, the new placement is biased to be similar to the original placement. The objective in Section 4 was to achieve run-to-run stability for inherently unstable placement algorithms. We use similar concepts to trade off predictability vs. optimization potential when rescaling a design. Our proposed flow for achieving predictability while rescaling a design is shown in Figure 7.

5.1 Experimental Flow

For our experiments, we use a well-known academic placer Capo [6] and the industrial placement tool QPlace from Cadence Silicon Ensemble(SEDSM). In the first step, initial placement of the design is obtained. In an actual scenario, this would be obtained from the layout optimized design that we are trying to rescale. In our experiments, we obtain this placement by running the placement tool on the design. We then re-floorplan the design to have a lower row-utilization (higher whitespace) compared to the original floorplan. A placer which uniformly distributes whitespace will ensure that the new floorplanned design has a higher local minimum whitespace compared to the original design. However, from our experiments we observe that the resulting new placements are in general not similar to the original placement in terms of timing characteristics. In most cases, the most critical paths of the new placement are totally different from the most critical paths in the original placement of the higher row-utilization design. This points to the instability in the placement algorithms we used. To achieve similarity in placement we use the following approach. We first rescale the locations of all the standard cells and terminals from the original placement to the new floorplan. This scaling is straight forward. Let the floorplan of the block scale from height h and width w to height h' and width w'. Then the location (x,y) of a standard cell in the original placement scales to the location (x',y') in new floorplan as follows.

$$x' = x * w'/w$$

$$y' = y * h'/h$$

After the rescaled placement has been obtained, the straight-forward thing to do is to apply ECO placement techniques to legalize the new re-scaled placement. This can be done efficiently using current ECO placement techniques such as QPlace run in the ECO mode. However, in a scenario where the netlist also changes considerably during the rescaling process, using local ECO changes will result in sub-optimal results. In our proposed flow (Figure 7), we tether the new rescaled placement using fake pins and fake nets as explained in Section 4. The new tethered netlist is then replaced. The tethering fake nets and pins ensure that the netlist is placed similarly to the original placement. We thus ensure

```
1  Rescale Placed Design
2    Obtain initial placement of the design;
3    Rescale locations of all standard cells and
     terminals to new layout dimensions
4    Tether x% of cells to resulting placement
5    Replace the tethered netlist
6    Remove the fake tethering pins and nets
```

Figure 7: **Proposed rescaling flow.**

Circuit	Function	#Cells	#Nets
AES	Rijndael core	10404	11955
MULT	53X53 Multiplier	13803	14359

Table 6: **Benchmarks used in our rescaling experiments. Advanced Encryption Standard/Rijndael (AES) core is encryption core downloaded from `http://www.opencores.org`. MULT is 53X53 bit multiplier that we synthesized using synopsys design foundation library.**

predictability during rescaling. After the placement, the fake pins and fake nets are removed from the netlist and this is the final re-scaled placement. The allowed freedom during the second placement run is governed by the number of standard-cells tethered and the region size of the tethering bounding box around each tethered standard-cell. Our proposed technique allows one to conveniently trade-off further optimization vs. predictability during the rescaling process.

5.2 Preserving Critical Paths

In the techniques presented in Section 4, the designer specifies the % of cells to be tethered to their initial locations. Specific cells are chosen at random. This approach works well when one is trying to focus mainly on the average similarity of the two placements. However, since in our case we are trying to preserve the timing behavior of the design during the rescaling process we select cells differently. From the initial layout-optimized design, we extract the cells in the top few (1000 in our experiments) worst paths in the design. We tether these critical cells to the locations, rescaled from their original locations. The remaining cells to be tethered are generated randomly. We thus try to ensure that the cells in the critical path are placed close to their rescaled locations with an aim to better preserve the timing behavior of the design.

As a variant of timing-driven tethering of cells, we also propose to change the size of the tethering region based on the dimensions of the critical nets in the original placement. By default, the tethering region is chosen to be a certain %(in our case 0.5% to 5%) of the layout dimensions. The size of the tethering region gives us a knob to trade off optimization vs. predictability. However, by tethering the critical cells to the bounding box of the critical net they belong to, one would maximize the optimization potential without sacrificing predictability in terms of timing behavior of the design.

Circuit	Row Util	TetherType	HPWL(e5)	Clk Period(ns)	# Similar paths
AES	70%	-	5.141	2.49	-
	50%	-	5.713	2.68	0/1000
	50%	Rand	6.211	2.87	**996/1000**
	50%	TD	6.138	2.68	**1000/1000**
MULT	70%	-	4.156	1.99	-
	50%	-	4.767	2.13	0/457
	50%	Rand	5.014	2.02	**279/457**
	50%	TD	5.002	2.07	**414/457**

Table 7: **Rescaling results for Capo. Tethered region size=0.5% of layout size.**

5.3 Experimental Results

Table 6 lists the characteristics of benchmarks used in our experiments. We downloaded the verilog code for the Advanced Encryption Standard/Rijndael (AES) core design from [18]. The MULT design is a 53X53 bit multiplier that we instantiated from the Synopsys design foundation library. We synthesized these designs using Synopsys Design Compiler and floorplanned them using Cadence Silicon Ensemble(SEDSM ver. 5.4). For our experiments we use the academic standard- cell placement tool Capo [6] and leading industrial placement tool QPlace from Cadence.

The results for rescaling are shown in Tables 7, 8 and 9. For all rescaling experiments, we first floorplan the die to have a row utilization of 70%. We run placement tools on the initial design to get a base placement and an initial timing report which is generated using Synopsys Primetime tool. All subsequent comparisons are made to this base placement. The design is then re-floorplanned to have row utilization of 50%. The netlist remains the same. We then replace the 50% utilized design. Next, we employ our proposed rescaling flow to maintain the timing characteristics of the design. For our flow, we choose 10% of the cells to be tethered for the second placement run. We apply two variants of our flow. In the first version we select the cells to be tethered randomly. The second version uses timing-driven tethering of cells by selecting cells on the top critical paths along with a few random cells, as the cells to be tethered. Table 7 presents the results for Capo when the size of tethering region around each tethered node was chosen to be 0.5% of the layout region size. As can be seen, without using tethering, Capo produces vastly different results in terms of tethering with 0 out of worst 1000 paths being similar for AES and MULT. We stabilize the placement considerably using our proposed flow and produce placements which have very similar timing characteristics compared to the original placement. However, the wirelength suffers around 10% due to tethering. One of the problems was that the tethered region size was too small, thus effecting the optimization potential of the placer. So we increased the area of the tethering region to 5% of the layout region. Results for this configuration of the experiment are shown in Table 8. With this change we are able to reduce the impact on HPWL to minimal while still maintaining the similarity in the timing behavior of the designs. We repeat the same experiment on rescaling using industry placer QPlace. Results presented in Table 9 show that QPlace, produces vastly different results in terms of timing behavior when the floorplan is changed a little. This result suggests that QPlace is using inherently unstable algorithms. Since our techniques mainly rely on pre-processing and post-processing of the input net-list, we can perform the same experiments on QPlace and try to improve its behavior in terms of predictability.

Circuit	Row Util	TetherType	HPWL(e5)	Clk Period(ns)	# Similar paths
AES	70%	none	5.141	2.49	-
	50%	none	5.713	2.68	0/1000
	50%	Rand	5.895	2.76	**997/1000**
	50%	TD	6.078	2.74	**1000/1000**
MULT	70%	none	4.156	1.99	-
	50%	none	4.767	2.13	0/457
	50%	Rand	4.869	2.08	**328/457**
	50%	TD	4.818	2.06	**403/457**

Table 8: **Rescaling results for Capo. Tethered region size=5% of layout size.**

Circuit	Row Util	TetherType	HPWL(e5)	Clk Period(ns)	# Similar paths
AES	70%	none	6.068	2.46	-
	50%	none	6.122	2.86	0/1000
	50%	Rand	6.591	2.48	**885/1000**
	50%	TD	6.732	2.88	**978/1000**
MULT	70%	none	3.821	2.26	-
	50%	none	4.295	2.19	25/457
	50%	Rand	4.776	2.11	**457/457**
	50%	TD	4.745	2.39	**457/457**

Table 9: **Rescaling results for QPlace. Tethered region size=5% of layout size.**

As seen from Table 9 we were able to produce placements with very similar timing characteristics even when we changed the row-utilizations of the design. However, the loss in HPWL due to tethering seems to be more significant for QPlace. We suspect this is because of the fake fixed pins introduced all over the layout during tethering. Capo does not reserve any space for these fake pins on the layout, however, QPlace seems to be reserving a site for each of these pins during placement, thus hurting the optimization of HPWL. Currently, we are trying to alleviate this problem by tweaking our QPlace flow.

6 Conclusions

Large-scale placement is becoming more sophisticated in the presence of large IP blocks, embedded memories and macros. Aggressive timing constraints, large whitespace and physical synthesis flows pose new challenges to layout tools. In particular, local and global incremental changes must be sustained without chaotic effects on congestion and circuit delay. We observe that "local" whitespace makes layouts amenable to local modifications and re-synthesis, while stability of placement results facilitates larger incremental changes.

We contribute simple and tunable techniques for ensuring minimum "local" whitespace throughout the core region without distributing all whitespace uniformly, and empirically demonstrate that such local whitespace is achieved with approximately 5% precision. Our study is complementary to that in [4] where whitespace is managed using a combination of min-cut and analytical placement techniques. Similarly, our methods can be used with congestion-driven whitespace

allocation from [21, 27]. Our empirical results show that lax controls over whitespace may lead to better half-perimeter wirelength, but at the same time may increase routed wirelength or even lead to unroutable designs. This may be the clearest example yet of the divergence between half-perimeter wirelength and routed wirelength as optimization objectives. Our experiments with physical synthesis point out that using a combination of free cells and uniform whitespace distribution during global placement can significantly improve circuit delay of low-utilization designs.

Our study of stability shows that while min-cut placers may produce solutions with very different congestion maps, it is possible to stabilize their results by a simple pre-processing. In fact, it takes a surprisingly small modification of the netlist to tie future placement solutions to a given set of locations. While some algorithms, e.g., analytical placement, tend to produce consistent results on multiple runs, our techniques can be used to tie the results produced by different placement algorithms and implementations to each other. In particular, placement predictions made by a fast estimator can be enforced at a global scale when a slower placer is used to optimize wirelength and various design objectives. We also address the issue of reshaping an existing layout-optimized design with an aim of preserving the timing characteristics of the design while still allowing room for further optimizations.

We apply our techniques for achieving stability in placers to devise a flow to rescale an existing layout-optimized design with the aim of preserving the timing characteristics of the design. We study the optimization vs. predictability trade-off in this context. The proposed rescaling flow is particularly useful when one also changes the netlist of the block during its re-implementation. Further, the rescaling flow is not limited to small changes in layout and netlist.

Straightforward implementations of the proposed techniques, such as free cells and fake nets, may increase the memory footprint of the placer and its runtime. Instead, those techniques can be implemented *implicitly* so as to guarantee the original memory footprint and only an insignificant slow-down. However, this is incompatible with the simple pre-processing approach that enabled our experiments with several placers.

Acknowledgments This work was supported by the Gigascale Silicon Research Center, an IBM University Partnership award and equipment grants from Intel and IBM. We would also like to thank Xiaojian Yang (Synplicity) and Amir Farrahi (Sun Microsystems) for technical discussions.

References

[1] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning Through Better Local Search", *ICCD* 2001, pp. 328-334.

[2] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-Blocks using Floorplanning and Standard-Cell Placement", *ISPD* 2002, pp. 12-17.

[3] S. N. Adya, M. Yildiz, I. L. Markov, P. G. Villarrubia, P. N. Parakh and P. H. Madden, "Benchmarking for Large-Scale Placement and Beyond," *in IEEE Trans. on CAD, vol. 23(4), April, 2004*, pp. 472-487.

[4] C. J. Alpert, G.-J. Nam and P. G. Villarrubia, "Free Space Management for Cut-Based Placement", *ICCAD* 2002, pp. 746-751.

[5] H. B. Bakoglu, “Interconnections and Packaging for VLSI”, Addison-Wesley, MA, 1990

[6] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?” *DAC* 2000, pp. 477-82.

[7] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Optimal Partitioners and End-case Placers for Standard-cell Layout”, *IEEE Trans. on CAD, vol. 19, no. 11, 2000*, pp. 1304-1314

[8] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Hierarchical Whitespace Allocation in Top-down Placement”, *IEEE Trans. on CAD, vol. 22, no. 11, 2003*, pp. 716-723

[9] A. B. Kahng and I. L. Markov, “VLSI CAD Bookshelf” `http://vlsicad.eecs.umich.edu/BK`

[10] C. C. Chang, J. Cong and M. Xie, “Optimality and Scalability Study of Existing Placement Algorithms,” *ASP DAC* 2003, pp. 621-627.

[11] C.-C. Chang, J. Cong, and X. Yuan, “Multi-level Placement for Large-Scale Mixed-Size IC Designs,” *ASPDAC* 2003, pp. 325-330.

[12] J. Cong, M. Romesis, M. Xie, “Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms”, *ISPD* 2003, pp. 88-94.

[13] J. Cong, “An Interconnect-centric Design Flow for Nanometer Technologies”, Proc. of the IEEE, 89(4), April, 2001, pp. 505-528

[14] W. J. Dally and A. Chang, “The Role of Custom Design in ASIC Chips”, *DAC* 2000, pp. 643-647.

[15] S. Dutt and H. Thenny, “Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxation”, *ICCAD* 1997, pp. 350-355

[16] H. Eisenmann and F. M. Johannes, “Generic Global Placement and Floorplanning”, *DAC* 1988, pp. 269-274.

[17] B. Goplen, P. Saxena and S. Sapatnekar, “Net Weighting to Reduce Repeater Counts during Placement”, *DAC* 2005, pp. 503-508

[18] `http://www.opencores.org`

[19] J. Lou, S. Krishnamoorthy, H. S. Sheng, “Estimating Routing Congestion using Probabilistic Analysis,” *ISPD 2001*, pp 112-117.

[20] P. N. Parakh, R. B. Brown, K. A. Sakallah, “Congestion Driven Quadratic Placement”, *DAC* 1998, pp. 275-278.

[21] A. Rohe and U. Brenner, “An Effective Congestion Driven Placement Framework,” *ISPD 2002*, pp. 6-11.

[22] M. Sarrafzadeh, M. Wang and X. Yang, “Modern Placement Techniques,” Kluwer 2002.

[23] P. Saxena, N. Menezes, P. Cocchini and D. Kirkpatrick, “The Scaling Challenge: Can Correct-By-Construction Design Help?”, *ISPD* 2003, pp. 51-58.

[24] N. Selvakkumaran, P. N. Parakh and G. Karypis, “Perimeter-degree: A priori Metric for Directly Measuring and Homogenizing Interconnection Complexity in Multilevel Placement”, *SLIP* 2003, pp. 53-59.

[25] L. Scheffer and E. Nequist, “Why Interconnect Prediction Doesn’t Work,” *SLIP* 2000, pp. 139-144.

[26] M. Wang, X. Yang and M. Sarrafzadeh, “Dragon2000: Standard-cell Placement Tool for Large Industry Circuits,” *ICCAD 2000*, pp. 260-263.

[27] X. Yang, B.-K. Choi and M. Sarrafzadeh, “Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement,” *ISPD 2002*, pp. 42-50.