On Whitespace and Stability in Mixed-Size Placement and Physical Synthesis

Saurabh N. Adya
University of Michigan
EECS Department
Ann Arbor, MI 48109-2122
sadya@eecs.umich.edu

Igor L. Markov
University of Michigan
EECS Department
Ann Arbor, MI 48109-2122
imarkov@eecs.umich.edu

Paul G. Villarrubia
IBM Corporation
11501 Burnet Road
Austin, TX 78758
pgvillar@us.ibm.com

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. Another requirement, the stability of placement results from run to run, is important to the convergence of physical synthesis loops. Indeed, logic resynthesis targetting 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.

In the context of earlier proposed techniques for mixed-size placement [2], we tune a state-of-the-art recursive bisection placer to better handle regular netlists that offer a convenient way to represent memories, datapaths and random-logic IP blocks. These modifications and better whitespace distribution improve results on recent mixed-size placement benchmarks.

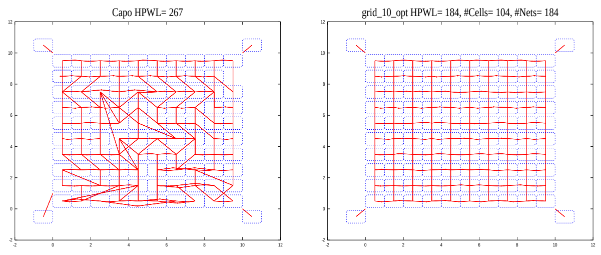
1. INTRODUCTION

With the rapid decrease of feature sizes circuit layouts become more complex, both in terms of size and design constraints [3]. Additional issues are brought up by massive IP reuse, embedded memories, the demand for more aggressive timing optimization and power density constraints. During circuit layout IP blocks and embedded memories often appear as large or medium-sized pre-designed black boxes that must be placed simultaneously with hundreds of thousands of small standard cells. This is commonly known as the mixed-size placement problem or the "boulders and dust problem". Our work enhances earlier proposed techniques in several ways. We modify the Capo placer [5] to better handle grid based, data-path style netlists. We also improve Capo performance on designs with limited whitespace using a two-phase partitioning approach.

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 [18] 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, researchers from IBM show [4] that distributing whitespace uniformly [7] may significantly increase wirelength. They also point out 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 [19]). 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 we 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 [14] of several min-cut placements and cell displacements. Surprisingly 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.

In the remainder, Section 2 gives background on large-scale placement and describes previous work. Our changes to a min-cut placer to perform better on regular layouts and in mixed-size placement are described in Section 3. Whitespace distribution is discussed in Section 4, mixed-size placement in Section 5 and stability in Section 6, accompanied by relevant empirical results. Our contributions are summarized in Section 7.



Capo 8.5 Capo 8.6

Figure 1: Capo placements for designs with regular grid connectivity. Capo 8.5 produces sub-optimal placements. Capo 8.6 produces the optimal placement for this design. There are 4 terminals connected to the 4 corner cells to anchor the design.

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 [5]. 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 caches similarly constraint random-logic partitions. In the context of massive IP reuse, especially with hard IP blocks, analog circuits (DACs, ADCs, PLLs) and embedded memories, the die area may be determined by floorplanning and area minimization during placement is typically 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 may allocate whitespace uniformly [5, 7] or according to congestion maps [16, 21]. 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 [18] 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 N-well contacts that have to be assigned to vertices of a grid and 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 [15], 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. 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 Mixed-size fixed-die placement

Mixed-size placement where large macros are placed simultaneously with numerous small cells (also known as the "boulders and dust" problem) becomes particularly complex in the fixed-die context because of its discreteness. Analytical mixed-size placers have been reported in [13] and other works, but they are typically validated on small variable-die layouts, and no competitive results are shown. Our experience indicates that such tools often place macros with significant overlap. Large publicly available mixed-size placement benchmarks have been proposed recently in [2], along with a design flow that produces placement solutions with zero overlap. Such results have been further improved in [10]. The reader is referred to the book [17] for a detailed background discussion of mixed-size placement. Below we summarize the work in [2, 10].

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 similar to Figure 1 right, 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.

While the shredding process is a key step, the methodology mostly relies on clustering, floorplanning and final re-placement. Thus the first step is mainly used to facilitate good clustering.

An entirely different approach is pursued in [10]. Their placer mPG-MS is based on an earlier tool mPG, which recursively clusters the netlist to build a hierarchy. The top-level netlist of approximately 500 clusters is placed using Simulated Annealing (SA), and then the placement is gradually refined by unclustering the netlist and improving the placement of smaller clusters by SA. mPG-MS contributes a structure of bins, in which large and small blocks are placed during coarse placement. The coarse placement is necessarily overlap-free for big objects, but small objects must be further replaced by a detail placer. A significant effort is expended to check for overlap during refinement and legalize possible violations.

Circuit	#Nodes	#Nets	WS %	Optimal HPWL	Dragon HPWL	Plato HPWL	Capo default HPWL	Capo + repart HPWL
10x10	100	184	0	184	293	202	267	184
95x95	9025	17864	5	17884	39687	18302	21828	22764
100x100	10000	19804	0	19804	46066	20519	38352	21314
190x190	36100	71824	5	71864	175623	75384	90665	89814
200x200	40000	79604	0	79604	198182	82335	193167	100041

Table 1: Wirelength achieved by several placers on regular grids of varying size and whitespace. Plato is the original implementation of KraftWerk by Eisenmann and Johannes. While Plato produces small wirelength on $n \times n$ grids, it often diverges on random-logic netlists with embedded grids.

3. BETTER PLACEMENT OF REGULAR NETLISTS

Observe that the mixed-size placement techniques from [2] call for placement of grid-graphs embedded into random-logic netlists. However, we discovered that Capo 8.5 placer used in [2] performs poorly on grid-graphs, as shown in Figure 1 which illustrates an optimal and a sub-optimal placement of a 10×10 grid with four fixed cells in the corners. This is hardly a surprise because generic standard-cell placers are known to perform badly on regular, datapath style designs [12]. Our improvements to Capo allow it to better handle regular netlists without the loss of performance on random-logic netlists. These improvements are described below, and their implementation was contributed to Capo 8.6.

3.1 A two-phase partitioning approach

During each partitioning step with a vertical cut line, Capo 8.5 with default parameters uses a fairly large tolerance (of the order of 10-20%) in order to find better cuts. After a good cut is found, the geometric cut line is adjusted according to the sizes of partitions, with an equal distribution of whitespace among the partitions. However, *if no whitespace is available in the block*, this technique can cause cell overlaps. Namely, since cutlines cannot cut through cell sites and since no "jagged" cutlines are allowed, the set of partition balances that can be realized with a straight vertical cutline and zero whitespace is fairly discrete. Capo 8.5 simply rounds the current balance to the closest realizable and sets the geometric cutline accordingly. When whitespace is scarce, one of the resulting partitions may be overfull and the other may have artificially-created whitespace. Only a relatively small number of cell overlaps can be created this way, but they can be spread through the whole core area. When used in the MetaPlacer shell, Capo 8.5 removes overlaps after global placement by a simple and very fast greedy heuristic which resolves overlaps at the cost of increased wirelength.

In an attempt to reduce the number of overlaps, we revise the partitioning process in Capo. When a placement block is partitioned with a vertical cutline, at first the tolerance is fairly large. This allows Capo to determine the location of the geometric cutline by rounding to the nearest site. Furthermore, if the block has very little whitespace, we then repartition it with a small tolerance in an attempt to rebalance the current partitions according to the newly defined geometric cutline. Such repartitioning may be less useful in placers with fixed cut-line, but the use of fixed-cutlines itself may increase wirelength.

3.2 Fuzzy terminal propagation

Another modification we implemented is related to terminal propagation in min-cut placers with moveable cut-lines. Normally, if a projection of a terminal's location is too close to the expected cutline, the terminal is ignored by Capo in an attempt to avoid excessively speculative decisions. The proximity threshold is defined in percent of the current block size, and this parameter is called "partition fuzziness". For example, suppose that the y location of a terminal is within 9% of the tentative location of the horizontal cutline. Then, with partition fuzziness of 10%, this terminal will be ignored during partitioning. Our studies of Capo performance on grids suggest that partition fuzziness should be tuned up, particularly for small blocks. For example, if a placement block has only three cell rows, then possible tentative locations of horizontal cutlines are relatively far from the center. In a neighboring block that has not been partitioned yet, all cells are "located" at the center of the block, causing all connected terminals to propagate into one partition in the current block. To avoid this, we increase partition fuzziness to 33%.

Our modifications are tested on the grid designs from [3]. The two changes described above improve the performance of Capo on the grid designs with 0% whitespace by a factor of two. Additionally, to test the performance of various available placement algorithms, the grid designs are placed by four different algorithms and the results are summarized in Table 1. Analytical placer KraftWerk [13] and the modified Capo placer [5] perform reasonably well on these netlists. Dragon [21] which combines recursive partitioning with simulated annealing doesn't do favorably on these purely regular designs. To validate our modifications, we also tested Capo8.6 on datapath designs from the Synopsys design foundation library. On a six-stage pipelined multiplier design with 10828 cells and 20% whitespace, routing a Capo placement produces a routed wirelength of $1.21e^9$ where as routing a Cadence QPlace placement produces a routed wirelength of $1.13e^9$. On a combined arithmetic and barrel shifter design with 1622 cells and 20% whitespace, routing a Capo placement produces a routed wirelength of $1.10e^8$ where as routing a Cadence QPlace placement produces a routed wirelength of $1.13e^8$. On the larger datapath design Capo8.6+WRoute flow is slightly worse than QPlace+WRoute flow. However, we point out that Capo8.6 does not explicitly take routability into account.

4. WHITESPACE MANAGEMENT

Min-cut placers that uniformly distribute whitespace [7] tend to produce excessive wirelength when large amounts of whitespace are present [4]. The authors of [4] propose a fairly sophisticated technique ACG that combines quadratic placement with min-cut. While we address the same problem, 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 placers. 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 cut-line to better reflect the outcome (balance) of every min-cut partitioning call, whereas the placer in [4] uses a grid of placement blocks rather than a more general slicing floorplan as in Capo. It has been argued that analytical or quadratic placement algorithms have a global view of the problem and can manage large amounts of whitespace better. Analytical Constraint Generation (ACG) [4] combines min-cut based placer with quadratic placement engine, to generate partitioning capacities during top-down recursive bisection based min-cut placement flow. According to [4], ACG technique manages whitespace better than a typical min-cut placer.

The technique we propose assumes a placer that uniformly distributes whitespace across the core area. We assume that the mini-

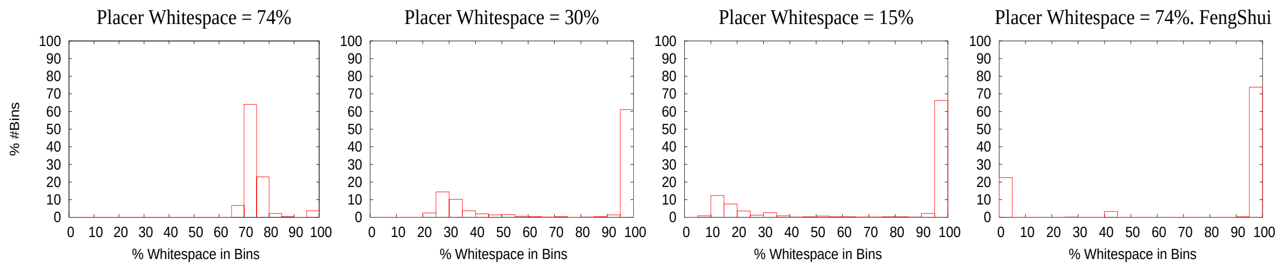


Figure 2: A histogram for local whitespace in design ckt4 from IBM with 74% whitespace. We subdivide the core area into a 27x27 grid and calculate local whitespace in each bin. We plot the % number of bins versus the % whitespace in each bin. Figure (A) shows the local whitespace distribution in a Capo placement with no filler 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 filler cells added to reduce the whitespace available to the placer. Filler cells are removed after placement for local whitespace computation.

	Capo 8.6						Dragon 2.23 (fi xed-die mode: -fd)					
%Filler Cells	Place WL(e8)	Place Time(s)	Routed WL(e8)	Route Time(s)	#Vio lations	Route Success	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 2: Place and Route results for `ibm02` benchmark from IBM-Dragon suite with whitespace increased to 65%. Filler 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.

mum “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 “filler cells” to the design in an amount not exceeding whitespace that remains after the “local” requirement is satisfied. Since filler cells are disconnected and small, a placer is free to place those cells so as to improve relevant design objectives. After placement, we remove filler cells and treat the remaining cell sites as empty. This causes high cell density in certain areas, with filler cells occupying the vacant areas of the chip.

Our empirical evaluation uses the Capo placer [5] which uniformly distributes available whitespace [7] with routability in mind (`Feng Shui 2.0` and `mPl 2.0` currently 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 bad timing. Figure 3 shows placements of an industrial design with 72940 nodes, 73155 nets and 74% whitespace. Figure 3 (A) shows the placement achieved by uniform distribution of whitespace and Figure (B) shows the placement achieved by introducing filler cells to reduce whitespace 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 filler 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. Filler 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 filler cells introduced during the placement, most of the bins have a local whitespace of around 70-80%. When filler 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 3 (B). However, most of the bins containing standard cells have a local whitespace around 30%. Similar effect is observed when filler cells are added to reduce whitespace to 15%. While we have not performed experiments with ACG, we suspect that it may further improve wirelength if used in conjunction with filler 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.

Physical synthesis flows interleave placement optimizations with logic optimizations to achieve desired timing on a design. 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, 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 filler cells on physical synthesis in Table 4. 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 filler 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

Circuit	Flow = Capo+Parquet+Capo [2] (High Temp Anneal)			Our Flow = Capo+Parquet+Capo (Low Temp Anneal)						mPG[10]	
	I			II			III			IV	
	Uniform WS			Uniform WS			Uniform WS + Filler Cells				
	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 3: 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 filler cells (II) and with filler 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].

minimum local whitespace of 40% around the core area. Filler cells are removed after global placement. As seen from the results in Table 4, the worst slack and total negative slack for all the designs improve considerably by adding filler cells during the global placement stage of physical synthesis. All the designs are routable even after compacting the designs by using filler cells.

We also conduct experiments to demonstrate the effect of filler cells on the routability of a design. We use the ibm02 benchmark from [21]. 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. Filler cells are gradually added during placement, reducing whitespace that the placer can allocate uniformly. Each of these designs is placed, then the filler cells are removed and the design is routed with Cadence WarpRoute. Table 2 reports the results of these experiments. Clearly, adding filler 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, filler cells are useful in reducing the half-perimeter wirelength, but distributing a portion of whitespace helps Capo produce routable placements. In fact, reporting 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.

5. IMPROVED MIXED-SIZE PLACEMENT

We show that better whitespace allocation reduces wirelength in mixed-size placement, and further improve the placement flow from [2] with unrelated techniques, including those from Section 3. Step 2 of that mixed-size placement flow forms clustered blocks of standard cells using physical clustering and uses a fixed-outline floorplanner to help remove overlaps between macros subject the fixed-outline constraints. Fixed-outline floorplanning is the main bottleneck in [2], mainly because satisfying a given outline takes a number of restarts. We speed-up this stage as follows. When forming soft clusters of standard cells using physical clustering, we reduce the area of each clustered soft block by 10%. Thus the area of the clustered block is 0.9 * (sum of areas of sub-cells). This increases the amount of whitespace available to the floorplanner and helps the fixed-outline floorplanner in finding a solution satisfying fixed-outline constraints faster. Additionally, instead of using full-blown annealing that starts with a random initial solution, we try to maintain the initial positions of macros (obtained from placing a shredded version of the netlist). This is done by forming a sequence pair from the illegal placement obtained from Step 1 of the flow and then employing low-temperature annealing.

Step 3 of the flow fixes the macro locations to the ones provided by the floorplanner and replaces standard-cells around the macros. Here we improve whitespace allocation by introducing filler cells. Figure 4 shows the improvements to the mixed-size placement flow. Figure 4 (A) shows the placement of ibm01 design after placing the shredded netlist. Figure 4 (B) shows the placement after floorplanning the design. Since low-temperature annealing is used, the macro locations follow the locations of Figure (A). Figure 4 (C) shows the design with macros fixed and standard cells placed around them with uniform whitespace distribution. We add filler cells to reduce the available whitespace to the placer to 10% and replace the design with the macros being fixed. Figure 4 (D) shows standard cells placed around macros with filler cells and uniform whitespace distribution. The filler cells are not shown. The results are summarized in Table 3. We compare our results to mPG [10].

6. STABILITY

Physical synthesis flows often require the stability of placement results from run to run for future optimizations targeting timing and/or congestion. However, Figures 5 (A) and (B) show that congestion maps [14] 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 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 lo-

Circuit	#Cells	no Filler Cells						w Filler Cells					
	(During	After Placement				After Phy-Synthesis		After Placement				After Phy-Synthesis	
	Placement Stage)	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 4: The impact of filler 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 filler cells so that the whitespace available to the placer was reduced to 40%. Filler cells are removed after global placement. All designs are routable after physical synthesis.

cations, and the amount of change allowed can be easily controlled during pre-processing. It is important to note that we are not adding 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.

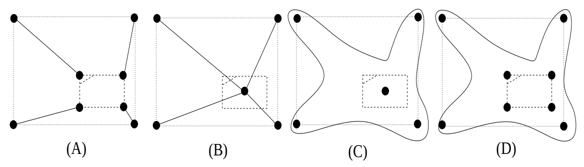


Figure 6: A single cell/macro is tied to a rectangular region in 4 different ways. Solid dots show artificially added (fake) pins, skew lines show fake two-pin nets, and a fake 5-pin net is shown by a spline. In all three cases moving the cell within the region does not affect the total length of fake nets. However, any placement beyond the region will incur a wirelength penalty that is independent of other movable objects. In (A), four fake pins are added in the corners to preserve cell orientation. In (B), one fake pin is added at the center so that changes in orientation do not affect wirelength. In (C), the same effect is achieved by using one fake 5-pin net rather than four fake two-pin nets. In (B) and (C), only the center of the cell is constrained to be in the region. In (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.

Figure 6 demonstrates several ways to tie a cell or a macro to a region without inducing a hard constraint. The four outer fake pins are fixed in the corners of the given region. Note that a technique similar to that in Figure 6 (A) is used in [2] to restrict orientations of macros (however, in that work the four outer fake pins are fixed at the corners of the core region). Three of the new constructions ignore 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 latter was suggested to us by Amir Farrahi from Synplicity and can be used to mitigate the number of added nets. The third new construction uses a single eight-pin net and can be used to ensure that the entire cell is placed within the constraining region and also reduces the number of added nets. Otherwise, these constructions are equivalent if used with min-cut placers or placers based on simulated annealing.

In our experiments, we randomly select 2%-5% cells in a given placement and tie them to regions centered at the cell's location. 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 5 (C), additional runs of the min-cut placer Capo produce essentially the same congestion map. The placement in Figure 5 (D) is tied to the output of Dragon. Table 6 reports the effect of tethering cells to a base placement on the IBM-v1 benchmarks [20]. 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 [20].[1] To stabilize such designs we randomly select for tethering at least one cell from each disconnected component in the netlist. Table 6 reports the average and maximum Manhattan difference between locations of nodes 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.[2] 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.

Table 5 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.

% 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 5: 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.

[1]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.

[2]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.

7. 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 resynthesis, 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 [16, 21]. 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 filler cells and uniform whitespace distribution during global placement can significantly improve circuit delay of low-utilization designs.

Our work improves the performance of state-of-the-art min-cut placer Capo on regular grid-like netlists and datapaths. This is particularly useful in a design flow where large macros are placed simultaneously with numerous small cells by means of shredding and subsequent legalization [2]. Our empirical results for mixed-size placement are significantly better than those reported in [2] and are comparable to those in [10]. It should be noted however, that the multi-level techniques in [10] are very different from those used by other researchers and can, in principle, be combined with ours or even applied to placements produced by our methods. We believe that even within our flow several additional improvements can be made, and our ongoing work pursues such possibilities.

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.

Straightforward implementations of the proposed techniques, such as filler 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. We do not pursue this avenue in our work because it requires non-trivial source code modification and 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 (Synplicity) for technical discussions, as well as Shubhyant Chaturvedi (The University of Michigan) for evaluating Capo on datapath circuits from the Synopsys design foundation library.

8. 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 et al., "Benchmarking for Large-Scale Placement and Beyond," *ISPD* 2003, pp. 95-103.

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

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

[6] 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

[7] 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

[8] A. E. Caldwell, A. B. Kahng, I. L. Markov, "VLSI CAD Bookshelf" `http://vlsicad.eecs.umich.edu/BK`

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

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

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

[12] W. J. Dally and A. Chang, "The Role of Custom Design in ASIC Chips", *DAC* 2000, p. 643-647.

[13] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *DAC* 1988, p. 269-274.

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

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

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

[17] M. Sarrafzadeh, M. Wang and X. Yang, "Modern Placement Techniques," Kluwer 2002.

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

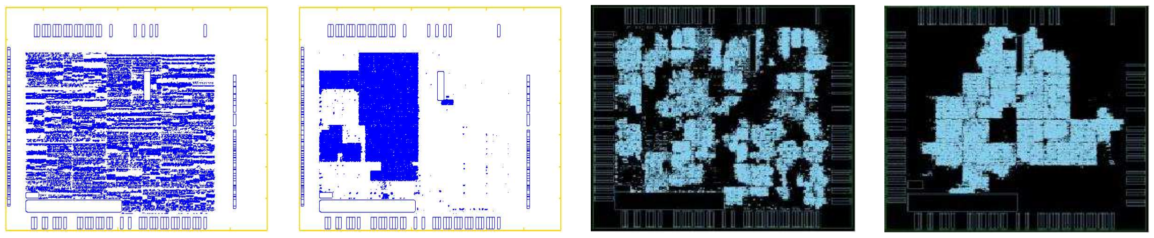
[19] L. Scheffer and E. Nequist, "Why interconnect prediction doesn't work," *SLIP* 2000, pp. 139-144.

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

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

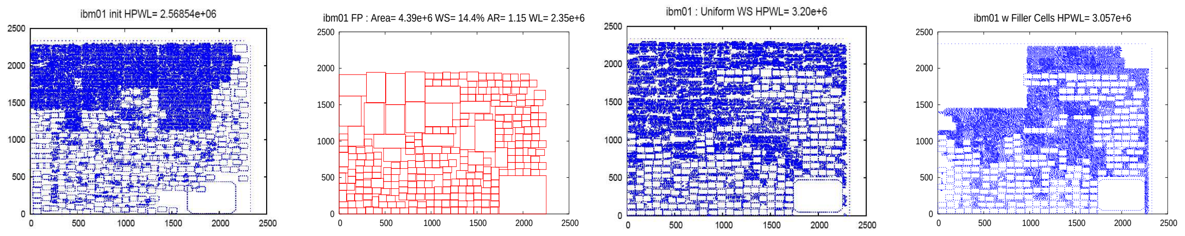
Circuit	#Cells	#Nets	No Tethering		2% Cells Tethered		5% Cells Tethered		10% Cells Tethered		50% Cells Tethered	
			%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff	%Avg Diff	%Max Diff
ibm01	12282	11507	46	98	6	38	2.6	32	3.1	38	1.1	39
ibm02	19321	18429	24	58	4.8	47	4	45	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 6: 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.



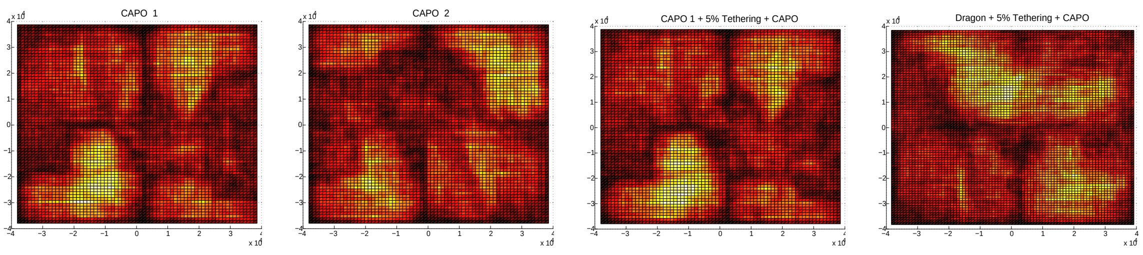
(A) (B) (C) (D)

Figure 3: The `ckt4` design from IBM has 72940 nodes, 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 filler cells were added to reduce placer whitespace from 74% to 15%. This reduces the half-perimeter wirelength from $15.32e6$ to $8.77e6$. Filler 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].



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

Figure 4: Placements of the `ibm01` design with 12752 nodes, 14111 nets, 246 macros and 20% whitespace. Figure (A) shows the placement obtained after placing the shredded netlist. Figure (B) shows the placement after floorplanning in low-temperature annealing mode to remove the overlaps between macros while trying to maintain initial locations of macros. Standard cells are clustered into soft blocks using physical clustering. Figure (C) shows standard cells placed around the fixed macros with uniform whitespace distribution. Figure (D) shows standard cells placed around the fixed macros after adding filler cells to decrease the whitespace to 10%. Filler cells are not shown in (D).



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

Figure 5: Placements of the `ibm02` design with 19321 nodes 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.