Free Space Management for Cut-Based Placement

Charles J. Alpert, Gi-Joon Nam and Paul G. Villarrubia
IBM Corporation, 11501 Burnet Road, Austin, TX 78758

Abstract

IP blocks and large macro cells are increasingly prevalent in physical design, actually causing an increase in the available free space for the dust logic. We observe that top-down placement based on recursive bisection with multilevel partitioning performs poorly on these porous designs. However, analytic solvers have the ability to find the natural distribution of cells in the layout. Consequently, we propose an enhancement to cut-based placement called Analytic Constraint Generation (ACG). ACG utilizes an analytic engine to set constraints for the multi-level partitioner. We show that for real industry designs, ACG significantly improves the performance of cut-based placement, as implemented within a state-of-the-art industrial placer.

1. Introduction

As design complexity continues to increase while time-to-market decreases, IP reuse and semi-hierarchical design are becoming increasingly pervasive. A few years ago, pure standard cell designs could be considered the norm, but today's "chunky" designs contain large blocks for memory arrays, IP blocks, etc. Consequently, today's placement instances now resemble the problem of arranging "dust" logics around the large blocks. Since the large blocks tend to dictate the design footprint, one can no longer assume that the total dust logic area matches the available free space in the design (minus the large blocks); one must recognize the trend of increasing percentage of "free space" available on the chip. One might think increased free space, or "design sparsity", might make placement easier. However, there can still be millions of dust logic cells that have profound effects on timing and routability.

As noted in [16], packing cells densely can yield the minimum wirelength solution, but create enough congestion to make the design unroutable. Uniformly spreading the design [8] may work well for dense design, but can unnecessarily hurt timing for sparse designs. We will show that one can control free space management to achieve better timing than via a uniform spreading strategy without hurting routability.

Placement algorithms are typically based on either a simulated annealing, top-down cut-based partitioning, or analytical [12] paradigm (or some combination thereof). Recent years have seen the emergence of several new academic placement tools, especially in the top-down partitioning and analytical domains.

The advent of multilevel partitioning [3][10] has helped spawn a new generation of top-down cut-based placers, e.g., [4][17][15]. A placer in this class partitions the cells into either two (bisection) or four (quadrisection) regions of the chip, then recursively partitions each region until a global coarse placement is achieved. As we will show, recursive cut-based placement can perform quite well when designs are dense but poorly when they are sparse. Sparse designs tend to fool the partitioner since it does not know how to allocate the large amount of free space to each sub-partition.

Analytic placers typically solve a relaxed placement formulation optimally, allowing cells to temporarily overlap [2]. Legalization can be achieved in several ways. The classic analytical placers [12][11] both use cut-based bipartitioning techniques to remove overlaps. Vygen's placer [14] uses a minimum movement based quadrisection algorithm. Eisenmann and Johannes [8] add forces that pull cells from high to low density regions and iteratively solve the formulation until cells are evenly distributed. The Mongrel placement tool [9] starts with a legal placement and iteratively moves cells to their ideal relaxed location, invoking a legalization procedure when an overlap occurs. The FMM placer [5] uses a "Fast Multipole Method" to resolve overlap constraints.

The question of whether analytic engines are even necessary given the advances in multilevel partitioning was posed in 1997 [3]. The argument is that analytic methods only serve to "seed" the partitioner since partitioning is typically used as a legalization step. Since then, several innovative legalization schemes [5][8][9][14] have been proposed that make this argument specious. Nevertheless, analytic placers can perform poorly when the data is naturally degenerate since it becomes difficult to legalize a placement where thousands of cells have virtually the same location. Also, analytic methods have difficulties with dense designs where legalization must significantly alter the analytic solution.

Given that designs are becoming sparser, we wish to utilize the strength of analytic placers on sparse designs to improve the poor performance of cut-based placers on these instances. We propose the Analytic Constraint Generation (ACG) technique that uses the solution of a quadratic analytic solver to set appropriate balance constraints for the multilevel partitioner. ACG serves to *guide* the partitioner into finding the natural distribution of cells (or free space) into sub-partitions. We present experiments on 1-dimensional and 2-dimensional instances that validate our claims of performance on sparse design and also show improved timing of ACG placements when used with a physical synthesis optimization engine.

2. Preliminaries

We first briefly overview the cut-based partitioning and analytic placement methods, assuming a 1-dimensional placement instance. One dimension allows us to focus more on the underlying technique and less on the actual implementation choices, especially for analytic legalization.

In 1-dimensional placement, we are given a row of potential cell locations, e.g., Fig 1. (a) shows an image with 16 possible cell locations with uniform cell sizes. Cells are to be assigned to one of the possible locations. In this example, assume that eight cells must be placed, which means the *design density* is 50%. The design density is defined as the total moveable cell area divided by the total available area in the placement region.

Recursive bisection iteratively assigns the cells into one

0-7803-7607-2/02/$17.00

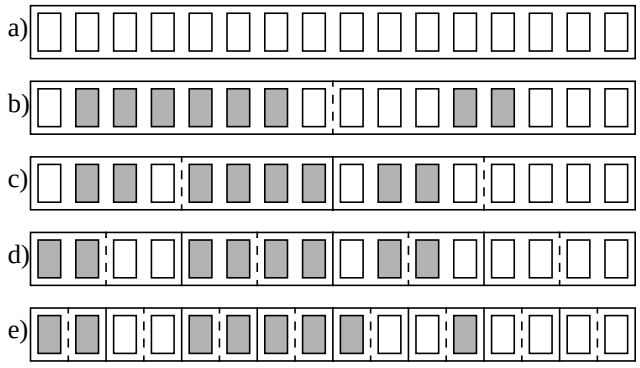

Fig 1. Recursive bisection on a linear placement problem with eight cells and 50% design density.

of two equal sized regions, though the cell area in the left region does not have to match the cell area in the right region. For example, in Fig 1. (b), six cells are assigned to the left region and two to the right. In Fig 1. (c)-(e), each smaller region is subsequently partitioned until there is only one cell or no cells in each region.

In analytical placement with a quadratic wirelength objective, one first solves the optimization problem

$$\phi(\vec{x}) = \sum_{i>j} w_{ij}(x_i - x_j)^2 \qquad (1)$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$ are the coordinates of the cells $v_1, v_2, \ldots, v_n$, some subset of the cell locations are fixed (e.g., pads), and w_{ij} is the weight of the net connecting v_i to v_j. This problem can be optimally solved with techniques such as successive-over relaxation or conjugate gradient methods. For the eight cell example in Fig 1., one typically finds the analytical optimization result looking something like Fig 2.. The cells may overlap significantly and form natural clusters that can be subdivided into the two regions left and right of the cut-line, as shown in Fig 1. (b). In subsequent recursive partitions, new constraints are added to the quadratic optimization, the system is solved again, and further partitioning is performed until all overlaps are removed.

Cut-based partitioning divides the cells into two partitions while trying to minimize the number of nets crossing partitions. The balance constraints determine how much flexibility the partitioner has to make cell assignments.

Let the *balance parameter* λ $(0 \le \lambda \le 1)$ be the desired ratio of the cell area of the left partition to the total cell area. If we want to divide the cells into two equal-sized parts, then $\lambda = 0.5$. However, if one side contains fixed cells or an odd number of rows need to be split, then λ could certainly take alternative values.

The *balance tolerance* Δ $(0 \le \Delta \le 1)$ is a parameter that specifies the amount of deviation that the partitioner is allowed to have. The balance tolerance is the sum of the upper and lower balance tolerances, Δ_{upper} and Δ_{lower} once a balance parameter λ is given. The maximum permissible value of Δ is the ratio of the difference between the maximum and minimum allowable cell areas of the left partition to the total cell area. For example, if no more than 40% and no less than 20% of the cell area is allowed to be assigned to the left partition then $\Delta = 0.2$. If the balance parameter λ is chosen to be 0.28, then $\Delta_{lower} = 0.08$ and $\Delta_{upper} = 0.12$. In general, the more free space is available in the design, the bigger balance tolerance is permissible.

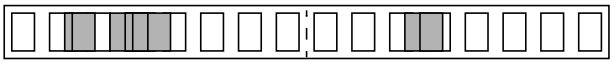

Fig 2. Example result of quadratic optimization on the Fig 1. example.

In top-down cut-based placement, one first divides the existing region into two (or possibly four) parts and commonly the relative sizes of these regions determines the value of λ. However, the relative sizes of the partitioning regions do not necessarily correspond to the desired partitioning ratio. Next, one examines how much free space there is for the cells to fit in these two regions. If all the total cell area equals the area of the region, then no free space is available and Δ must be set to zero (though a small value can be useful so that large cells are not prevented from changing partitions). Otherwise, Δ is typically set to the largest value that still yields a feasible solution.

In Fig 1. (a), the first cut could possibly have assigned all eight cells to either the left or right partition, which means Δ could be set to one for the first cut. However, in the second cut of the left partition (Fig 1. (b)), six cells need to be partitioned into two regions, each with four available cell locations, and Δ has a maximum value of $1/3$. If one chooses to give the bipartitioning algorithm the maximum possible flexibility, then Δ equals the ratio of available free space to total cell area. The partitioner is constrained to assign cells to regions such that ratio of the area of the left partition to total cell area must lie between $\lambda - \Delta_{lower}$ and $\lambda + \Delta_{upper}$.

The choice for λ can significantly affect solution quality. Consider the four cell example in Fig 3. (a) with no available free space; hence, we must have $\Delta = 0$. When $\lambda = 0.6$, the partitioner has the ability to find the min-cut solution which assigns A and B to one side and C and D to

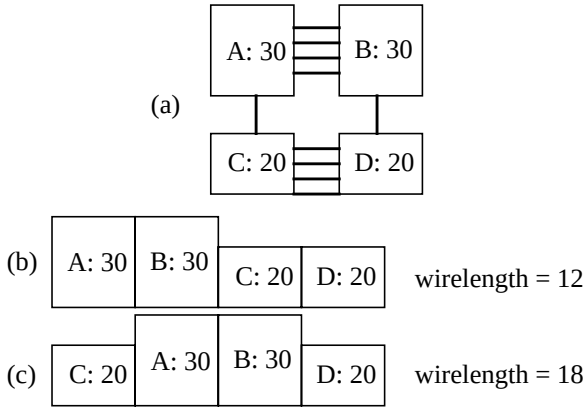

Fig 3. For the (a) 4-node circuit, using (b) $\lambda = 0.6$ yields the optimal placement but (c) $\lambda = 0.5$ yields a worse result.

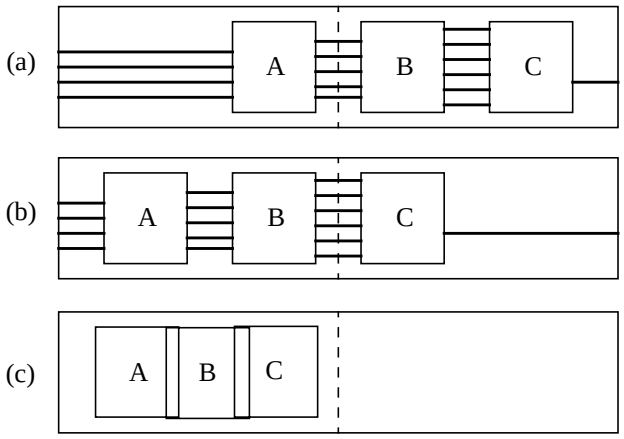

Fig 4. An example with three cells and one unit of free space. The (a) min-cut solution results in a placement with more wirelength than (b) the optimal solution.

the other, eventually yielding the minimum wirelength solution of 12 as shown in Fig 3. (b) (assuming cell width one and wiring to the center of each cell). A value of $\lambda = 0.5$, however, causes the minimum cut to jump from two to eight, resulting in the inferior result with wirelength 18 shown in Fig 3. (c).

This example suggests that one should allow the partitioner to try a range of possible λ values to find the min-cut solution. This could be accomplished by using a large value for Δ, allowing region area constraints to be violated by the partitioner. The region sizes themselves could then be adjusted so that they meet the area constraints of the partitioning solution, which in effect yields a new λ value.

However, allowing large variations in early cuts could overly constrain the partitioner, which leads to poor solutions for subsequent iterations. This is the classic weakness of greedy algorithms. In Fig 1., if the internal connectivity of the cells dominates connectivity to the pads, then the optimal greedy solution for the first cut assigns all eight cells either entirely to the left or the right partition. All subsequent iterations would then have no available free space which can yield sub-optimal solutions.

Finally, cut-based partitioners can suffer from their inability to see the global picture. Fig 4. shows an example with three cells and one potential cell of free space. The first partitioning must assign one cell to one side and two cells to the other. The min-cut solution in (a) results in a placement with more total wirelength than (b) the solution with a slightly higher net cut. Note that an analytic solver would place all three cells to the left of the cut-line, as shown in (c). Just about any legalization scheme for this solution would also obtain the result in (b).

3. Analytic Constraint Generation (ACG)

We have seen that an analytic solver can have a better global view than a cut-based placer, especially for sparse designs. Rather than discard the entire top-down cut-based placement methodology, we propose to utilize the strength of analytical solvers within a cut-based placer. We call our algorithm to do this Analytic Constraint Generation (ACG)

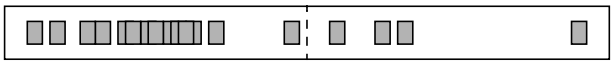

Fig 5. Example unbalanced analytic optimization.

since we use an analytic solver to compute the balance parameter that constrains the multilevel partitioner.

Instead of trying to adjust Δ to give the partitioner more freedom of choice, we set Δ to zero and try to find the best value a priori of the balance parameter λ. For example, Fig 5. shows a possible analytic optimization where a majority of cells are to the left of the partition. A 50/50 cut is possible, but a more unbalanced cut such as 80/20 ($\lambda = 0.8$) might exploit the unconstrained wirelength optimization better.

The idea of ACG is to use the result of the squared wirelength optimization to choose the balance parameter. Assume the analytic optimization yields the solution $\vec{x} = [x_1, x_2, \ldots, x_n]$, where these horizontal coordinates of the n cells are all free to move (A vertical optimization should be performed for a horizontal cut-line). Let

$$x_\mu = \frac{1}{n}\sum_{i=1}^{n} a(v_i) \cdot x_i \qquad (2)$$

be the *center-of-mass* of the solution, where $a(v_i)$ is the area of cell v_i. Let $\alpha = \sum a(v_i)$ be the total moveable cell area, and assume we wish to partition the cells into two rectangular regions A and B. For any rectangle R, let $a(R)$, $w(R)$ and $h(R)$ denote the area, width, and height of R, respectively. For the cells to fit in the regions, we must have $\alpha \le a(A) + a(B)$.

The width of the entire partitioning region is given by $w(A) + w(B)$. We construct a new rectangle R with the same aspect ratio as the partitioning region, but with area α, the total moveable cell area. The height and width of this new rectangle are given by

$$h(R) = \sqrt{\frac{\alpha \cdot h(A)}{w(A) + w(B)}} \text{ and}$$
$$w(R) = \sqrt{\frac{\alpha(w(A) + w(B))}{h(A)}} \qquad (3)$$

respectively. The rectangle R is placed so that its center on the horizontal axis is x_μ.

Fig 6. shows an example where the ratio of cell area α to region area $a(A) + a(B)$ is 0.5. In (a), the solution to the quadratic wirelength optimization is shown, where the center-of-mass is significantly to the left of the cut-line. In (b), the new rectangle with area α is shown centered at x_μ. Let x_c be the coordinate of the cut-line. Note that the cut-line divides the new rectangle into two unbalanced regions. It is the relative areas of these new regions that are used to compute λ as follows.

The widths of these new left and right rectangles C and D are given by

$$w(C) = \frac{w(R)}{2} + x_c - x_\mu \text{ and } w(D) = \frac{w(R)}{2} - x_c + x_\mu \qquad (4)$$

respectively. The ratio of the area of C to R is equal to the ratio of their widths, since both C and R have the same

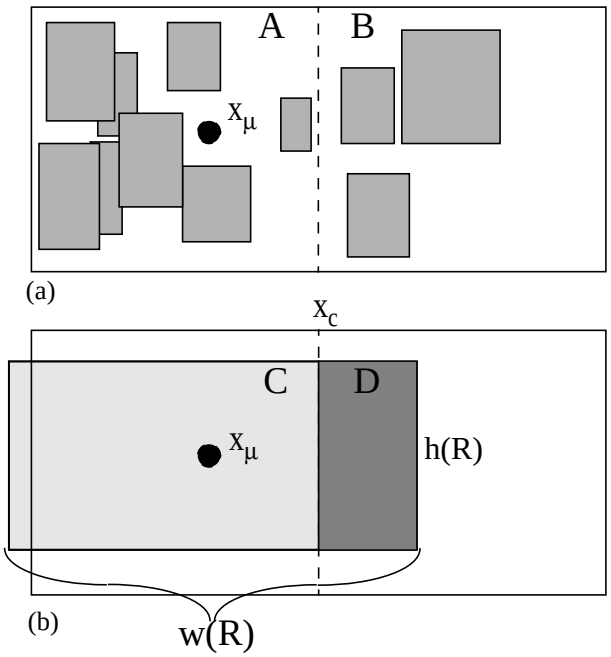

Fig 6. Example of the ACG algorithm for computing λ.

height. Thus, ACG uses the following value of λ:

$$\lambda = \frac{w(C)}{w(R)} = \frac{1}{2} + \frac{x_c - x_\mu}{w(R)} \quad (5)$$

There are two special cases that have to be considered. First, if x_μ is extremely far to the left or the right, then Equation (5) could produce a value for λ outside of its acceptable range. If $\lambda < 0$, we use a value of zero for λ, i.e., all cells are assigned to the right partition. Similarly, if $\lambda > 1$, we use a value of one.

Second, the condition might arise that either $a(C) > a(A)$ or $a(D) > a(B)$, in which case the chosen value of λ will cause an overflow of either A or B. In this case, we slide the rectangle R horizontally towards the cut-line until both $a(C) \le a(A)$ and $a(B) \le a(D)$ hold. Then, x_μ is modified to be the new horizontal center of R and Equation (5) is reapplied.

4. Experiments

We perform two sets of experiments. The first examines multilevel cut-based placement, analytical quadratic optimization, and ACG for 1-dimensional instances. The purpose is to illustrate the behavior of these different approaches for the same designs with various *degrees of sparsity*. The second set analyzes cut-based placement with and without ACG for a set of real industry circuits.

4.1 Linear Placement

Our first experiment utilizes the ISPD98 benchmark suite [1] that were transformed into standard cell placement instances by the authors of [15]. For each benchmark, we create a single circuit row such that the area of the row is equal to 20 times that of the total cell area, giving the design a density (total cell area divided by total placeable area) of

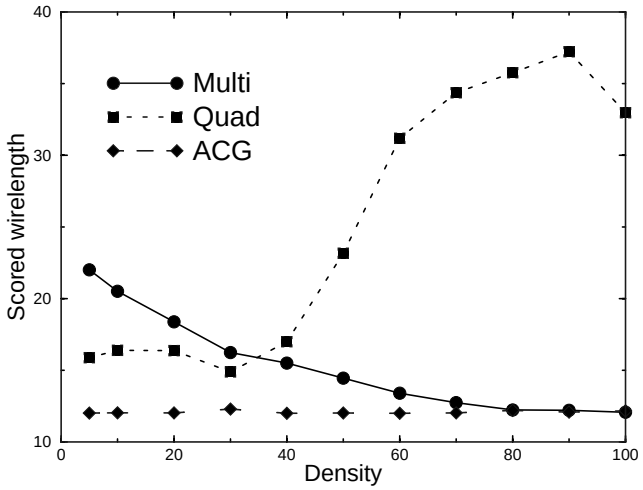

Fig 7. Total wirelength score as a function of density.

5%. The density of each design could be altered by artificially inflating the cell size; in this manner, we can generate instances for densities ranging from 5% to 100% for each design. The horizontal coordinates for the fixed I/O pads were spread out proportionally to span the entire row. The vertical coordinates were not altered. We ran three algorithms:

- **Multi**: recursive bisection for multilevel partitioning. For each partition, we used $\lambda = 0.5$ and the largest possible value for Δ.
- **Quad**: quadratic wirelength optimization. At each iteration, Equation (1) is solved and each partitioning region is divided into two equal parts by a vertical cut-line. If either partition overflows its area constraint, cells are moved to the non-overflowing partition, prioritized from distance to the cut-line, until both partitions satisfy area constraints.
- **ACG**: just like Multi, except using the algorithm from Section 3 to compute λ and a Δ value of zero.

For each design and density, the total wire lengths of the three algorithms were compared. Then, the wirelength for a given algorithm is divided by the smallest wirelength of the three algorithms to yield the score. For example, ibm03 with 5% density yielded wirelength of 1.16×10^9, 1.75×10^9, and 2.10×10^9 for ACG, Quad, and Multi, respectively. Since 1.16×10^9 is the smallest wirelength, ACG receives a score of 1, Quad a score of 1.51, and Multi a score of 1.81. Summing these scores over 12 benchmark circuits produces the graph in Fig 7..

One can clearly observe the following trends.

- For sparse designs, Quad outperforms Multi, though neither approach is as strong as ACG.
- Multi performance improves relative to ACG as designs become denser until the two algorithms virtually converge on design densities of 80% and up.
- For design densities of 50% and above, Quad results start to degrade considerably.

Overall, for lower densities ACG dominates both Multi and Quad. The reason that Quad performs relatively well for

small densities is that it excels at finding the general region where the cells belong. For example, in a design with 5% density, Quad typically compacts all the cells into the same narrow range of the circuit row, while Multi tends to spreads out the design. However, once Quad finds the right 5% of the design in which to pack the cells, it does a poor job of ordering the cells within that space (which also explains its poor performance for dense designs). Meanwhile, ACG succeeds at finding the same 5% region of the chip to place the cells as Quad but does a much better job at ordering them within that region. In that sense, ACG behaves like Quad during partitioning when free space is abundant, but like Multi when free space is in short supply.

4.2 Placement on Complete Designs

Our second experiment seeks to compare Multi and ACG on a set of seven real industry designs. We do not compare to an analytical method because (i) different legalization schemes behave very differently for 2-dimensional instances, and (ii) our purpose is to see how ACG enhances cut-based placement, not to attempt to demonstrate superiority over purely analytic placement methods.

We implemented both ACG and Multi within the CPlace [13] placement tool. CPlace has been used in the design and production of hundreds of ASIC parts and several microprocessors. For experiment, we ran the following flow on each test case:

- CPlace with multilevel partitioning with (labeled ACG) and without (labeled Multi) the ACG algorithm
- The PDS [6] physical synthesis tool which attempts to improve timing via buffer insertion, gate sizing, pin swapping, local logic changes, etc.
- The HDP global router which is typically used for congestion estimation by industry designers

Cplace was run using a *density target* of 70%, which prohibits the CPlace partitioner from packing any local region with more than 70% of the cells.[1] The density target serves to force cells to be spread out enough to avoid locally congested unroutable regions. The timing optimization is for reducing the worst slack value and the number of negative timing slack paths. Table 1 summarize the results. The reported statistics are:

- *Cells*, the number of moveable objects of design;
- *Design density*, the ratio of placeable cell area to total available space in the placement region;
- *TWL*, the total half-perimeter wire length in centimeters;
- *Worst slack*, the slack of the slowest path in the design in nano seconds;
- *FOM (Figure of Merit)*, a measure of the cumulative slack of all negative slack cells in the design (It can also be interpreted as the amount of work left for the designer. The closer the value is to zero, the better the overall timing characteristics of the design are);
- *# Neg. Paths*, the number of paths with negative slack;
- *Congestion Metric*, an average congestion of the worst (most congested) 20% of nets. A value lower than 80 indicates the design is likely routable;
- and *CPU* is the total placement runtime (only Cplace) on an IBM RS/6000 260 machine with 2Gb of RAM.

We make the following observations:

- ACG returns a better wire length before PDS than Multi, for all but the two densest designs, ckt 2 and ckt7. Given that these densities are close to the target density, we would not expect ACG to perform particularly well in these cases (as seen in our 1-dimensional experiments).
- After PDS, the timing characteristics for ACG are significantly better than Multi. The worst slack is better for every case except ckt7, and the FOM for ACG, the broadest measure design quality in terms of timing, is better than for Multi for all test cases. Thus, total wire length does not necessarily give a fair indication of the quality of the design in terms of satisfying timing constraints.
- The wire congestion results for ACG are higher than for Multi, but all the designs are below the 80 threshold of routability
- ACG uses about 28% more CPU time than Multi. Note that both our implementations of ACG and Multi successfully place fairly large designs in just a few hours.

Fig 9. and Fig 8. illustrate the placements of both Multi and ACG on ckt4. One can see that ACG is able to reduce the total wire length from 11.42 to 10.38 by packing the cells more tightly together than Multi. The design can be packed even further; if one uses a target density of 100%, then Multi and ACG obtain wire lengths of 10.78 and 9.60, respectively. Thus, an entirely different challenge becomes trying to determine the appropriate target density so that the design is still routable. We generally observe that higher target densities lead to smaller wire lengths and even larger gaps between the performance of Multi and ACG. This behavior is expected as the sparsity of the design effectively increases with the target density.

5. Conclusion

We have shown the amount of free space and the methodology to distribute it has a significant impact on placement. Cut-based approaches perform poorly compared to analytical placement on sparse designs. To remedy this shortcoming, we proposed ACG, a technique to generate partitioning constraints for a cut-based placer. Experimental results show that ACG significantly improves the performance of an industrial multilevel cut-based placement tool, especially in terms of timing. We believe that there are ample opportunities to further improve existing placement technology for the domain of sparse, chunky designs.

References

[1] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite", *Intl. Symposium on Physical Design*,1998, pp. 80-85.

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

[3] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel Circuit Parti-

[1] This constraint can be implemented by artificially inflating cells, e.g., for a 70% target density, cells are inflated by a factor or 1.43. Of course, one must take care not to overflow the design if the design density is greater than the target density.

Test Case	Cells	Design density	Method	Before PDS		After PDS				Congestion Estimation	CPU
				TWL	Worst Slack	TWL	Worst Slack	FOM	# Negative Paths		
ckt1	207K	65%	Multi	68.88	-6.30	68.37	-3.02	-6009	6162	70.52	5205
			ACG	67.47	-6.77	67.28	-2.57	-4671	7240	77.04	6022
ckt2	71K	73%	Multi	13.42	-2.44	13.59	-0.31	-173	1416	73.85	1414
			ACG	13.81	-2.11	13.99	-0.24	-71	811	76.32	1589
ckt3	120K	53%	Multi	136.54	-190.49	138.80	-3.25	-26870	25739	63.16	3118
			ACG	133.33	-29.45	133.68	-2.37	-7321	13687	70.43	4091
ckt4	73K	31%	Multi	11.42	-2.82	11.65	-0.82	-106	253	73.74	930
			ACG	10.38	-2.69	10.64	-0.57	-69	244	77.89	2096
ckt5	270K	45%	Multi	92.87	-7.12	92.40	-2.31	-9905	19549	63.36	3580
			ACG	84.27	-5.30	84.08	-1.93	-8617	11389	69.10	6508
ckt6	426K	57%	Multi	222.01	-5.10	232.85	-0.16	-2	78	69.87	11906
			ACG	220.82	-5.28	230.88	0.00	0	0	73.64	13774
ckt7	276K	69%	Multi	214.21	-8.39	214.21	-2.23	-10169	15233	69.15	7479
			ACG	214.29	-8.58	224.28	-4.51	-6582	8784	75.49	8921

Table 1: Comparison of Multi to ACG for real industry circuits.

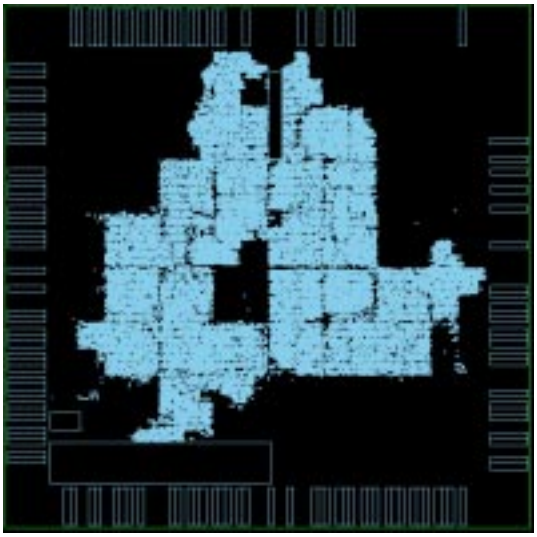

Fig 8. ACG placement of ckt4 with 70% target density.

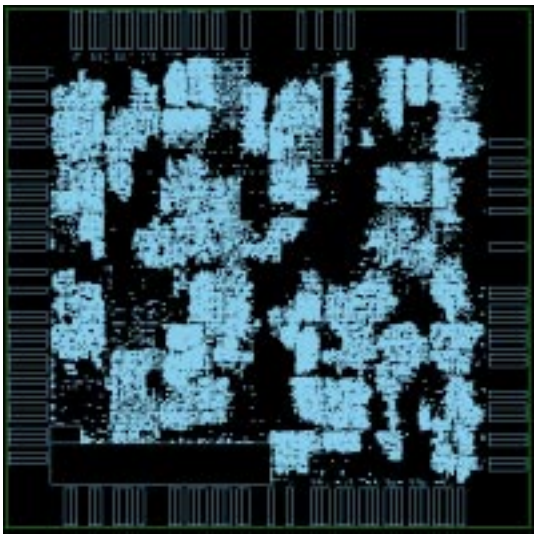

Fig 9. Multi placement of ckt4 with 70% target density.

tioning", *IEEE/ACM DAC*, 1997, pp. 530-533.

[4] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements", *IEEE/ACM DAC*, 2000, pp. 477-482.

[5] T. F. Chan, J. Cong, T. Kong, and J. R. Shinner, "Multilevel Optimization for Large-Scale Circuit Placement", *IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2000, pp. 171-176.

[6] W. Donath, P. Kuvda, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty, "Transformational Placement and Synthesis", *Design Automation & Test in Europe*, 2000, pp., 194-201.

[7] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques", *IEEE/ACM Intl. Conf on Computer-Aided Design*, 1996, pp. 194-200.

[8] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *IEEE/ACM DAC*, 1998, pp. 269-274.

[9] S.-W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement", *IEEE/ACM ICCAD*, 2000, pp. 165-170.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain", *IEEE/ACM DAC*, 1997, pp. 526-529.

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

[12] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu, "PROUD: A Fast Sea-of-Gates Placement Algorithm", *IEEE/ACM DAC*, 1988, pp. 318-323.

[13] P. Villarrubia, G. Nusbaum, R. Masleid, and P. T. Patel, "IBM RISC Chip Design Methodology", *ICCD*, 1989, pp. 143-147.

[14] J. Vygen, "Algorithms for Large-Scale Flat Placement", *Proc. 34th IEEE/ACM Design Automation Conference*, 1997, pp. 746-751.

[15] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Standard-Cell Placement Tool for Large Industry Circuits", *IEEE/ACM Intl. Conf on Computer-Aided Design*, 2001, pp. 260-263.

[16] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement", *International Symposium on Physical Design*, 2002, pp. 42-47.

[17] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement", *IEEE/ACM DAC*, 2001, pp. 776-779.