Algorithms for Detailed Placement of Standard Cells

Jens Vygen
Research Institute for Discrete Mathematics
University of Bonn
Lennéstr. 2, 53113 Bonn, Germany

Abstract

The state–of–the–art methods for the placement of large–scale standard cell designs work in a top–down fashion. After some iterations, where more and more detailed placement information is obtained, a final procedure for finding a legal placement is needed. This paper presents a new method for this final task, based on efficient algorithms from combinatorial optimization.

1 Introduction

The flat design of today's high performance processor chips requires a program for placing several hundred thousand objects with respect to wiring and timing constraints. In this paper, the objects to be placed will be called *circuits* (other authors speak of modules, cells or components). We assume that most of them are *standard cell circuits*; those have some standard height H and an integral width between 1 and W. A legal placement arranges the standard cell circuits in *cell rows* which can be assumed to be given in advance. From the placement point of view, gate–array designs also fit into this model.

All successful placement algorithms for the most complex designs work in a top–down fashion (see e.g. [1], [4], [5]). The global structure is always similar: Iteratively the set of circuits is partitioned into smaller and smaller subsets, each corresponding to a region of the chip area. These regions are induced by a finer and finer grid. A more detailed description will be found in Section 2.

After some iterations the regions are quite small; each of them contains only few circuits. When the regions' sizes become comparable to the size of the larger circuits the partitioning process is stopped. Now some final legalization procedure is needed. Most circuits can usually be placed within the region they are assigned to, but in some cases modifications will be necessary. This final phase is called *detailed placement* and is the subject of this paper.

We propose a new algorithm for the detailed placement. This new method naturally partitions the detailed placement task into several subtasks. Each of the subproblems can be formulated as a well–known combinatorial optimization problem and is solved optimally.

The detailed placement algorithm presented here consists of three phases: Balancing the regions (Section 3), balancing the zones (Section 4) and zone–based placement (Section 5). Some concluding remarks make up the final section.

2 Top–down placement

In this section we briefly review the global structure of top–down placement algorithms like [1], [4], [5]. This is necessary since the output of the global placement procedure is the input for the detailed placement.

Let $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ be the chip area. At each stage of the global placement we have two lists of coordinates $x_{\min} = x_0 < x_1 < x_2 < \cdots < x_{p-1} < x_p = x_{\max}$ and $y_{\min} = y_0 < y_1 < y_2 < \cdots < y_{q-1} < y_q = y_{\max}$ which define a grid, partitioning the chip area into $p \cdot q$ regions. By the region (i, j) we mean $[x_{i-1}, x_i] \times [y_{i-1}, y_i]$.

Let C denote the set of movable circuits. At any stage of the global placement, the set of circuits is distributed to the regions, i.e. we have an assignment $f : C \to \{1, \ldots, p\} \times \{1, \ldots, q\}$. Initially we of course have $p = q = 1$ and $f(c) = (1, 1)$ for all $c \in C$.

Now each iteration consists of the following steps. First of all, the grid is refined. That is, new cut coordinates (either horizontal or vertical or both) are generated and thereby each old region (i, j) is partitioned into two or four new regions. (In late stages, some of the regions might not be partitioned anymore if the final number of regions is not a power of 2.)

When a region (i, j) is partitioned, the set of circuits c with $f(c) = (i, j)$ has to be partitioned, too. Each of these circuits has to be assigned to one of the two or four subregions. After this one proceeds with the next iteration.

The assigment of the circuits to subregions is obviously the crucial step. The idea is that if a circuit is assigned to region (i, j), it will (later) probably be placed within this region. So a constraint to be observed is that the total size of the circuits assigned to one region does not exceed the region's capacity.

By capacity we mean the total usable area of the region. For example, if some circuits are fixed in advance, the area occupied by them cannot be used anymore. Furthermore, it is often desirable that a certain percentage of each region remains free, either for later use or for wirability reasons. This may also reduce capacity of a region.

At a late stage, each region contains only few circuits. Then an assignment of the circuits to the subregions meeting the capacity constraints might not exist. For example, if a region contains three circuits of size 10 and each of the two subregions has capacity 15, there is no feasible assignment. At this point it is recommendable to stop the iterative partitioning process.

What follows is the detailed placement phase. We assume that the rows of the final grid are exactly the cell rows. The columns of the final grid should have some reasonable size, depending on the distribution of the sizes of the standard cell circuits.

For example, Table 1 shows the distribution of the widths of the 198226 movable standard cell circuits of the MBA chip described in [2]. Here a column width of 50 is appropriate, as more than 99% of the circuits are smaller.

Table 1: Widths of standard cell circuits of a chip

width	#circuits
1	2655
2	27700
3	27822
4	23065
5	21562
6	20299
7	10789
8	26881
9	2897
10	1549
11	2888
12	1848
13	2160
14	172
15	274
16	222
17	12428
18	7661
19	1228
20	973
21-30	1112
31-50	1098
51-100	798
100-123	145

All circuits which do not fit into one region (in particular all circuits whose height exceeds the height of a cell row) are assumed to be fixed before the detailed placement starts.

3 Balancing the regions

Although the partitioning is stopped when the regions reach a certain size, it will typically happen that some capacity constraints are violated. However, if one region's capacity is exceeded there should be an adjacent region (or at least one not too far away) whose capacity is not used up. Thus a balancing of the regions should be possible without moving circuits too far. This is the goal of the first phase of the detailed placement algorithm.

First, a directed graph G is computed. The vertices of G are the regions; adjacent regions are joined by a pair of oppositely directed edges. The cost $c(r, r')$ of a directed edge (r, r') is a rough estimation of the cost of moving one unit, i.e. a circuit of width one, from region r to region r'. This does not mean that region r actually contains a circuit of width one: Rather we shall measure the cost of moving circuits of total width w from r to r' by $w \cdot c(r, r')$. We implemented the easiest possible estimation, namely a uniform cost for all horizontal edges and another one for all vertical edges (the ratio depends on the ratio of row height to column width). In any case, the edge costs should be nonnegative.

Let $w(r)$ be the width of region r, and let $C(r)$ be the set of the circuits currently assigned to region r. For any subset of circuits C' we write $w(C')$ for their total width. Moreover, suppose that the desired density is d; e.g. $d = .8$ means that preferably only 80% of a region should be used. Then the region's capacity is computed as $H \cdot w(r) \cdot d$. If all capacity constraints during the partitioning would have been met, we would have $H \cdot w(C(r)) \leq H \cdot w(r) \cdot d$ for all regions r.

Now we compute a number $b(r)$ for each region r, indicating how much free capacity is left or otherwise how far the capacity is exceeded. We set

$$b(r) := \begin{cases} w(r) \cdot d - w(C(r)) & \text{if } w(r) \cdot d > w(C(r)) \\ w(r) - w(C(r)) & \text{if } w(r) < w(C(r)) \\ 0 & \text{otherwise} \end{cases}$$

In the first case we call r a *sink*, in the second case r is a *source*. The above definition of b means that, as long as their total width does not exceed the width of the region, we don't force circuits to move, even if the local density is higher than d. This reflects that the density constraint is soft, it is not necessary to satisfy it for every single region.

Now a minimum–cost flow problem is set up. We introduce a super–source s, a super–sink t, edges (s, r) for all sources r, and edges (r, t) for all sinks r. The new edges have zero cost. The capacity of an edge (s, r) is $-b(r)$, the capacity of an edge (r, t) is $b(t)$, all other edges have infinite capacity. Then a maximum s–t–flow of minimum cost is determined. This can be done e.g. with Orlin's algorithm [3] in $O(n^2 \log^2 n)$ time, where n is the number of regions.

Now that an optimum flow f has been determined, it has to be realized by actually moving circuits. Let G' be the subgraph of G containing only the edges carrying positive flow. Since the edge costs are nonnegative, G' is an acyclic graph. We scan the vertices in topological order with respect to G', i.e. for any edge (r, r') with $f(r, r') > 0$ we examine r before r'. Such a topological ordering can be determined easily in linear time.

When examining a vertex r, we check if $w(C(r)) \leq w(r)$. In this case we do nothing. Otherwise it will be guaranteed that the total flow on the edges leaving r will be at least $w(C(r)) - w(r)$. Let $(r, r_1), \ldots, (r, r_k)$ be the outgoing edges with positive flow. We look for a partition $C(r) = C_0 \cup C_1 \cup \cdots \cup C_k$ into disjoint subsets such that

(i) $w(C_i) \leq f(r, r_i)$ for $i = 1, \ldots, k$;

(ii) $\max\{0, w(C(0)) - w(r)\}$ is minimum;

(iii) in case of a tie, the total cost $\displaystyle\sum_{i=1}^{k} \sum_{c \in C_k} cost(c, r, r_k)$ is minimum.

If the partition is found, we move C_i to r_i ($i = 1, \ldots, k$) and leave the circuits of C_0 in region r.

Objective (ii) reflects the fact that one cannot always guarantee $w(C(0)) \leq w(r)$. In such a case the capacity violation should be kept as small as possible. In (iii), $cost(c, r, r_k)$ is the cost of moving circuit c from r to r'. As a natural cost measure we take the increase of the (weighted) bounding–box netlength. Moreover, it is reasonable to add an extra cost if the circuit c has already been been moved quite far by previous steps.

It remains to show how to find an optimum partition. This problem turns out to be a kind of knapsack problem and can be solved efficiently by dynamic programming techniques:

Let $C(r) = \{c_1, \ldots, c_m\}$. For $j = 1, \ldots, m$ and $0 \leq x_i \leq f(r, r_i)$, $i = 1, \ldots, k$, we define $P_j(x_1, \ldots, x_k) := (C_1, \ldots, C_k)$, where $w(C_i) = x_i$, $C_i \subseteq \{c_1, \ldots, c_j\}$ ($i = 1, \ldots, k$) and $C_i \cap C_j = \emptyset$ ($i \neq j$), and among the k–tuples satisfiying these conditions $\displaystyle\sum_{i=1}^{k} \sum_{c \in C_k} cost(c, r, r_k)$ is minimum. If no such k–tuple exist, $P_j(x_1, \ldots, x_k)$ is undefined.

By subsequently scanning the circuits in the order $c_1, \ldots, c_m$ we can, for each $j = 1, \ldots, m$, compute $P_j(x_1, \ldots, x_k)$ for all $0 \leq x_i \leq f(r, r_i)$. This takes $\displaystyle\prod_{i=1}^{k} f(r, r_i)$ comparisons. Therefore the total running time is $O\left(\dfrac{w(C(r))^{k+1}}{k^k}\right)$, which is acceptable since $w(C(r))$ is not too big (less than 100) and k is very small (typically one or two, never more than four).

In practice the overall time for solving all these problems has proven to be neglectable compared to the time the minimum–cost flow computation requires. Note that in the special case $k = 1$ the above reduces to a standard knapsack problem.

4 Balancing the zones

The second phase of the detailed placement algorithm considers the so–called *zones*. By a zone we mean a maximal part of a cell row which is not intersected by any preplaced object or blockage. The goal of the second step is to guarantee that no zone contains more circuits than fit into it. In contrast to the previous region balancing, even slight capacity violations cannot be tolerated.

Basically the same framework as in the first phase is used, with the difference that the regions are no longer the ones induced by the grid. Another difference is that subsequent zones in the same cell row are called adjacent, although they have a positive distance. The cost of the edges of course reflects this distance.

An initial assignment of the circuits to the zones is obtained in a straightforward way, based on the final assignment to the regions (after the region balancing) and on the locations of the circuits.

Since now a zone can be adjacent to more than four other zones, the k in the dynamic programming algorithm for realizing the flow may now be too big. In such a (rare) case simple knapsack problems are solved for one outgoing edge after the other.

If, after finding an optimum flow and realizing it, there are still zones r for which $w(C(r)) > w(r)$, the procedure is iterated. We included two additional heuristics to ensure that after a few iterations (usually after the first one) the process terminates: First, for a source r, we increase $-b(r)$. Namely, we choose $-b(r)$ to be the smallest integer t greater than $w(C(r)) - w(r)$ such that a subset $C' \subseteq C(r)$ with $w(C') = t$ exists. Second, for all sinks r for which $b(r)$ is smaller than the largest $|b(r')|$ of a source r' we redefine $b(r) := 0$. The meaning of these two heuristics is easy to understand.

One may ask whether phase one (Section 3) is necessary at all. It isn't. However, experiments showed that the total movement of circuits is smaller if phase one is included. Also the netlength is usually better. A reason is that, due to very wide zones, realizing even a small flow in phase two can result in moving some circuits quite far.

5 Zone–based placement

After phase two it is guaranteed that $w(C(r)) \leq w(r)$ for all zones r, i.e. no zone contains more circuits than fit into it. Now all the circuits will be placed disjointly within their zones. Furthermore, the horizontal ordering of the circuits within each zone is fixed according to the locations of their centers. Observing these constraints, the optimum placement with respect to the (weighted) bounding–box netlength can be found.

Namely, the problem can be formulated as a linear program (LP). To do this, we need some notations. For each zone i we denote by $x^i_{\min}$ and $x^i_{\max}$ its left and right boundary. Let $c^i_1, \ldots, c^i_{m_i}$ be the circuits in zone i, in this horizontal order. For each circuit c, we introduce a variable $x(c)$ representing the x–coordinate of the center of the circuit. (Note that the

y–coordinate is fixed at this stage.)

Let N be the set of nets. Each net n is a set of pins. Each pin p belongs to a circuit $c(p)$ and has a fixed offset $(xoffs(p), yoffs(p))$ to the center of the circuit. Furthermore, each net n has a weight $w(n)$, reflecting its timing criticality. For each net n, we introduce two variables $\underline{x}(n), \overline{x}(n)$ representing the left and right boundary of the bounding box of n.

Using these notations, the LP formulation is:

$$\begin{aligned}
\min \quad & \sum_{n\in N} w(n)\,(\overline{x}(n) - \underline{x}(n)) & \\
\text{s.t.} \quad & \underline{x}(n) \le x(c(p)) + xoffs(p) \le \overline{x}(n) & \\
& & \text{(for all } p \in n) \\
& x^i_{\min} \le x(c^i_1) & \text{(for all } i) \\
& x(c^i_j) + \frac{w(\{c^i_j, c^i_{j+1}\})}{2} \le x(c^i_{j+1}) & \\
& & \text{(for all } i, j) \\
& x(c^i_{m_i}) \le x^i_{\max} & \text{(for all } i)
\end{aligned}$$

This LP is the dual of a minimum–cost flow problem. To see this, we introduce an auxiliary variable v_0 with the value zero. Then every constraint has the form $v_i - v_j \ge a_{ij}$, where a_{ij} is a constant and v_i and v_j are variables. Each of these constraints corresponds to a directed edge in a graph and to a dual variable f_{ij}. As the dual LP we then obtain

$$\begin{aligned}
\max \quad & \sum_{i,j} a_{ij} f_{ij} & \\
\text{s.t.} \quad & \sum_i f_{ij} - \sum_k f_{jk} = \begin{cases} -w(n) & \text{if } v_j \text{ is } \underline{x}(n) \\ w(n) & \text{if } v_j \text{ is } \overline{x}(n) \\ 0 & \text{otherwise} \end{cases} & \\
& & \text{(for all } j) \\
& f_{ij} \ge 0 & \text{(for all } i, j)
\end{aligned}$$

This is obviously a maximum–cost flow problem with infinite capacities. With the fastest known algorithm for uncapacitated minimum–cost flow problems [3] we can thus solve both the primal and the dual LP in $O((n \log n)(m + n \log n)$ time, where $n = |C| + |N|$ and m is the total number of pins.

For very large chips, this superquadratic running time is unacceptable. Therefore one may compute the optimum placement for each zone separately. As a consequence, one does not achieve the global optimum; the netlength is typically one or two percent higher.

We finally address the question how the density constraints can be taken into account in the zone–based placement. This is easy: By adding constraints of the type

$$x(c^i_j) + \frac{w(\{c^i_j, c^i_k\})}{2} + \sum_{l=j+1}^{k-1} w(\{c^i_l\}) + \delta \le x(c^i_k),$$

where $j < k$ and $\delta > 0$, it is ensured that at least δ units of free space will be somewhere between the circuits c_j and c_k. It should be clear how the density constraints can be translated into constraints of the above type. Of course, the efficient solvability is unaffected.

In certain circumstances it might be preferable to find a detailed placement with minimum total movement instead of minimum netlength. By considering artificial nets connecting a circuit's old x–coordinate to its center, this objective function can be viewed as a special case of the above. Moreover, it can be achieved that no single circuit is moved too far.

6 Concluding remarks

An additional postoptimization routine has been implemented. This procedure tries to find favourable exchanges of groups of circuits placed in adjacent zones. However, we found that the netlength gain of this local improvement heuristic is very small and usually not worth the computation time. This observation may also be viewed as an indication of the quality of the placement. All experiments are based on the placement program described in [5].

The detailed placement algorithm presented here has been used very successfully, e.g. for the design of the chips of the IBM S/390 Parallel Enterprise Server – Generation 3 (see [2]) and its successors. The running time for the MBA chip with 198226 movable standard cell circuits is about half an hour on an IBM RS/6000 595, depending on the density constraints imposed. As could be expected, the minimum–cost flow computations in phase one and three dominate the running time.

I thank Christoph Albrecht, Ursula Bünnagel, Martina Fuhrmann and Mathias Hauptmann for implementing the algorithms described here.

References

[1] J. M. Kleinhans, G. Sigl, F. M. Johannes, K. J. Antreich : GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems* 10 (1991), 356-365

[2] J. Koehl, U. Baur, B. Kick, T. Ludwig, T. Pflueger : A Flat, Timing–Driven Design System for a High–Performance CMOS Processor Chipset, *this volume*

[3] J. B. Orlin : A Faster Strongly Polynomial Minimum Cost Flow Algorithm, *Operations Research* 41 (1993), 338-350

[4] R.-S. Tsay, E. Kuh, C.-P. Hsu : Proud: A Sea–Of–Gate Placement Algorithm, *IEEE Design and Test of Computers*, 1988, 44-56

[5] J. Vygen : Algorithms for Large–Scale Flat Placement, *Proceedings of the 34th Design Automation Conference*, 1997, 746-751