A Standard-Cell Placement Tool for Designs with High Row Utilization

Abstract

In this paper we study the correlation between wirelength and routability for standard-cell placement problem, under the fixed-die place-and-route environment. We present a placement tool with better routability for designs with high row utilization. Compared to a well-known industrial placement tool, our placer produces placements with equal or better routability, 13.2% better half-perimeter wirelength, 15.3% better routed wirelength, and 9.1% less vias. Compared to a state-of-the-art academic placement tool Capo, our placer produces placements with significantly better routability, 14.5% better half-perimeter wirelength, 18.1% better routed wirelength, and 8.2% less vias. Experimental results show that purely minimizing wirelength, without congestion optimization, still improves routability and layout quality. Several novel algorithmic details are presented in the paper with experimental results. The framework and detailed implementation of our placer are described and various placement techniques are investigated. Additionally, we have built a set of benchmarks with reasonable circuit sizes and standard-cell sizes.

1. Introduction

Standard-cell placement problem has drawn extensive research attention in VLSI CAD area ever since its appearance. Nowadays, in spite of wide use of hierarchical design methodology and floorplanning, the problem of placing standard-cells remains one of the important topics. This problem is becoming more challenging because of two reasons. First, the circuit sizes are growing dramatically. The Semiconductor Research Corporation (SRC) suggests that we should be able to place designs containing up to one million cells within 16 hours. Such a problem size means that there exists a huge space between the optimal solution and the best solution by any known heuristic. The second reason is the multi-objective placement process. In addition to the traditional objects such as routability and timing, more issues must be taken into account during the placement, e.g., cross talk, power. The placement problem becomes harder when considering these objectives.

One common classification for traditional placement methods is to put them into four basic categories: min-cut placement [1, 2], simulated annealing [3], analytical method [4, 5], and force-directed approach [6]. However, recently proposed placement tools rarely reside in any one of these categories. Most of them are more or less hybrid models[1]. Four classical techniques, plus clustering and flow-based method, frequently appear in these relatively new placement algorithms [10, 11, 12, 13, 14, 15, 16, 17, 18]. In addition to the above approaches that address half-perimeter wirelength, many techniques are proposed for timing [19, 20, 12, 21, 22, 23] and congestion [24, 25, 26] optimization. Most of them are based on wirelength minimization.

Wirelength is the fundamental objective in standard-cell placement problem. It is generally believed that a timing or congestion oriented approach can hardly be successful without a good wirelength minimization engine. The idea of timing driven placement is to reduce the wirelengths on certain paths instead of the total wirelength. A placement with shorter total wirelength is relatively easier to be modified to meet timing constraints. Similarly, a good placement with optimized wirelength has a higher probability that its congested regions are relatively smaller or less serious.

The major contributions of this paper are the following. First, we study the correlation between half-perimeter wirelength and routability under practical physical design environment (using practical designs and an industrial router). We claim that the desire to shorten half-perimeter wirelength would never hurt although there does exist a mismatch between wirelength and routability. Second, we present a state-of-the-art placement tool with superior quality on both wirelength and routability. For designs with high row utilization, this tool produces placements with better routability compared to a well-known commercial tool. Finally, we describe several novel algorithmic details and many implementation elements of the proposed placement tool. As a side effect of this work, we have built a set of benchmarks for routability study which we believe are more relevant than current placement benchmarks in the academic research community.

The remainder of this paper is organized as follows. In Section 2, we describe the work background and environment. We then introduce the framework of our new standard-cell placement tool in Section 3. In Section 4, we explain the relevant aspects of implementation, and present novel algorithmic details. Section 5 shows the superior quality of the placement tool by comparing it with an industrial tool. We conclude and discuss future works in Section 6.

2. Background

2.1 Routability

For standard-cell placement, in order to study routability problem with high credibility, a good router is necessary. In this work we use a mature industrial router to evaluate placement quality.

In addition to the success/failure of the routing, the number of violations is an essential indicator of the routability. Another measure of routability is the routing time. In our experiments we often observe big difference (sometimes one is 5 or 6 times longer than another) on routing time for two placements, even though both routings are successful. Shorter routing time means less iterations on rip-up and re-route to fix the violations, thus corresponds to better routability.

Total routed wirelength and the number of vias are two measurements of layout quality (not routability) if the routing is successful. They are not very useful if the routing fails, but still good indicators for evaluating placement quality.

2.2 Benchmarks

Collecting benchmarks is an inevitable step of this work. The old MCNC benchmarks are small (except one circuit). Recent IBM-PLACE benchmarks [14] are in reasonable size, but lack physical information. We scale circuits in IBM-PLACE to match the standard-cell sizes in a $0.18\mu m$ library, which was obtained from Artisan Components Inc. through the academic research support program. We then output a pair of LEF/DEF files and use an industrial floorplanner to decide the core size and rows. Using an industrial place-and-route tool as the standard, we found the *transitional* benchmarks by changing the number of routing tracks between rows, and the number of routing layers.

The above benchmarks do not have power, ground and clock net. To acquire real designs, we downloaded ISPD01 benchmarks [22]. These benchmarks are in structural verilog file format. We use an industrial synthesis tool to compile them with $0.18\mu m$ standard-cell library, and then perform the same procedure as described above. After the procedure we have benchmarks of real designs.

2.3 Variable-die and fixed-die

A good description of *fixed-die* and *variable-die* can be found in [7]. Most previous placement techniques were developed under the variable-

[1]Some exceptions are [7, 8] (pure min-cut) and [9] (analytical, with flow method in detailed placement).

die assumption[2], while fixed-die is the common style in the real design world. Fortunately, the gap is not as big as it looks, because most placement techniques do not care whether variable-die or fixed-die is used. Nevertheless, the difference does exist. One obvious example is: in fixed-die placement with large white space, placing all the cells close to each other will get better wirelength. But the congestion will be worse than a spread-out placement. The trade-off between congestion and other metrics (wirelength, delay) must be made. Also it is very hard for placement tool to handle congestion without knowledge of routing tracks, as an excessive congestion estimate leads to more efforts on congestion minimization, impairing wirelength and delay.

Handling congestion in fixed-die mode is beyond the scope of this paper. We deliberately choose very small white space for all benchmarks. For these benchmarks, there is no difference between variable-die and fixed-die approaches. We want to understand the wirelength and routability problem in this context, and then move forward to the placement with typical fixed-die mode, i.e., larger white space. Specifically, we set white spaces of all the benchmarks less than 2%, corresponding the row utilization more than 98%. We believe that under this high utilization circumstance, the study on the wirelength and routability will help us understand the correlation between them[3].

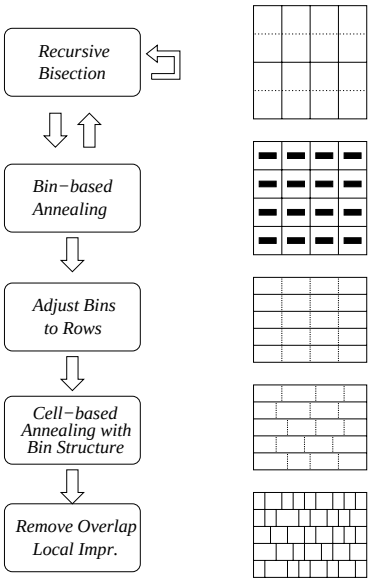

Fig. 1: Framework of Our Placement Tool

3. Framework of Our Placement Tool

3.1 Overview

The flow of our placement tool is shown in Fig. 1. This is a similar flow as in [14]. Compared to the work in [14], our placement flow contains different features which will be described in Section 4. We also address some practical problems on which [14] does not give the answer. These problems include the limitation on the number of rows and balancing the lengths of rows.

The main placement flow consists of two parts: recursive bisection and simulated annealing. These two techniques appeared very early in the literature of standard-cell placement and have proven effective. The recent advances in multilevel partitioning [27, 28] and their implementations imposed effects on the placement research in academia. Several placement tools [7, 14, 8] were proposed based on them.

As Fig. 1 shows, the given circuit is recursively partitioned along alternatively horizontal and vertical cut line. The subcircuits after partitioning are assigned to rectangular bins. At some points a bin-based simulated annealing (i.e. the moving objects are the subcircuits in the bins) is performed to improve the current placement. Such a procedure terminates when certain stop criteria (e.g. average number of cells per bin is less than a given number) are satisfied. An adjustment step is then executed to fit the current bin-based placement into row structures. The next step is a cell-based simulated annealing. The bin structure still exists and the cells are moved between the centers of bins. The locations of these centers can be changed during the annealing procedure. After that, the final step simply spreads overlapped cells, and makes local improvement to obtain the detailed placement.

3.2 Advantages and Disadvantages

Implementation

Multilevel partitioning implementation is available in source code or C library files. In our case we use *hMetis* [27] as the partitioning tool. Also, the cooling schedule of simulated annealing is well-known. Compared to analytical or flow-based approaches, this flow is easier to be implemented.

[2]There were a few placement papers presenting experimental results on fixed-die mode, including [7] and [22]

[3]The benchmarks in [7] have various white space ratio up to 30%. In this case, the half-perimeter wirelength in placement is no longer a good indicator of routability.

Simulated Annealing

A weakness of pure min-cut type placement is its irreversibility: once a cell is assigned to one side of the cut line, it will never move to the other side to improve the placement. Combining simulated annealing in this flow helps placements move out of the local minima.

We use *multilevel simulated annealing* in this placement flow. It is similar to the hierarchical annealing used in [29]. The key idea is to reduce the number of movable objectives in annealing. The difference between our flow and [29] is: instead of using a single cooling schedule through three hierarchical levels, we use low temperature annealing at each level and do not fix the number of levels. Moreover, we avoid using simulated annealing at final placement stage and use a fast greedy improvement instead. Our final placement step takes approximately 5% of the total runtime, while the final annealing stage in [29] often takes more than 70% of the total runtime.

Both bin annealing and cell annealing use total wirelength as the cost function. Also they adopt the same cooling schedule. Swapping is the main move in both types of annealing and shifting is lightly used in cell annealing (discussed in Section 4.5).

The disadvantage of simulated annealing is its expensive runtime cost. Although our flow tries to reduce this cost by bin-based approach, annealing is still the most time consuming part.

Bin Based Approach

Bin based placement is widely used [11, 14, 17]. The advantage is that its regular structure provides various ways to speed up computationally expensive operations in placement. At the cell annealing step in our flow, bin structure confines the change of every move to be local. This is much faster compared to the old "flat" annealing, in which every cell move or swap changes locations of many other cells.

The weakness of bin based method is that it requires that the cells are equally divided into bins. Therefore the tolerance (or unbalance factor) of partitioning has to be small. This often degrades the quality of partitioning. Another drawback is that the bin based method can hardly be extended to handle big macro cells.

There are two additional problems caused by bin-based structure: the limitation on number of rows and bin unbalance. They are solved by two new approaches proposed in this work. Please see Section 4.3 and Section 4.4 for details.

3.3 Data Structure

In addition to cells, models and nets, pin structure is indispensable. Pins of the same cell (or net) should be placed together in memory for higher efficiency[4]. The other net (or cell) to pin mapping could be implemented by a linked list.

Bin structure contains a double linked list for all cells in the bin. This is for cell annealing within bin structure. Like the linked list for pins, the linked list for bins can be built statically, as every cell belongs to one and only one bin.

Two data structures can be used to speed up the bin annealing step. A new net list could be created by eliminating the same nets or an external net list could be built for every bin (internal nets are useless). We use the second method in the implementation.

Each net has two fields other than the basic data. One is to save the current bounding box before calculating new value. Thus a failure move trial in annealing does not need computing bounding box again. The other field is an integer tag indicating that the net was already visited when swapping two cells or cell groups.

It is very useful, at the final placement stage, to sort the cells within the same row and build a double linked list for each row. The reason is that we need to quickly find neighbors of a given cell at local improvement step.

4. Detailed Implementations

In this section, we present details of implementation for this placement flow. Due to the space limitation, not every detail is showed with support of experiment data. However, all of them are real problems we have encountered and every conclusion comes from many experimental runs.

4.1 Cut Minimization and Wirelength

Cut minimization is the basis of min-cut type placements. It is also the key factor in our hybrid placement flow. The quality of the partitioning affects the final wirelength result substantially.

However, if we compare a min-cut placer and our placer by analyzing the net-cut sequence for the final placements, opposite phenomenon is showing. The placement produced by a min-cut placer has better net-cut at the first several levels than our placer, while our placer outputs placement with shorter total wirelength. Fig. 2 illustrates this observation.

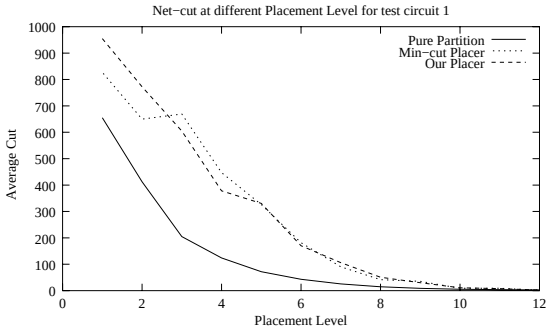

Fig. 2: Net-cuts at different placement levels. The curve of the placement by our placer corresponds higher net-cuts at the coarser placement levels, compared to the placement by min-cut approach. The curve of "Pure Partition" indicates the net cuts obtained by recursive bi-partitioning without terminal propagation.

This suggests that, at earlier levels of our placement flow, the cut minimization does not have to be greedy. In other words, we only need to do a reasonable job for early partitionings. Runtime cost can be reduced without much loss of quality. Experimental results showed in Table 1 support this point. The total runtime of partitioning can save up to 88% without much loss in the quality of final placement.

[4]This is because the increment (by one) operation is faster than an add operation in most machines.

test circuit 1			test circuit 2		
level	*wirelength*	*runtime*	*level*	*wirelength*	*runtime*
1	4.300	130	1	4.524	130
2	4.273	124	2	4.544	120
3	4.324	119	3	4.561	110
4	4.317	112	4	4.549	96
5	4.361	97	5	4.598	70
6	4.347	65	6	4.539	40
7	4.368	52	7	4.560	16

Table 1: Wirelength and runtime comparison for placement using different partitioning effort. *level* indicates at which placement level we start using high-quality partitioning (longer runtime). Before that level the partitioning is in the fast mode (worse quality). Wirelengths are in meters. Runtime is total partitioning runtime in seconds. Every entry is the average value from three placement runs.

test circuit 1		test circuit 2	
level	*wirelength*	*level*	*wirelength*
1	4.316	1	4.560
2	4.328	2	4.521
3	4.322	3	4.608
4	4.264	4	4.508
5	4.305	5	4.503
6	4.311	6	4.582
7	4.329	7	4.492

Table 2: Wirelength and runtime comparison for placement using different terminal propagation strategies in partitioning. *level* indicates at which placement level we start using terminal propagation in partitioning. Wirelengths are in meters. Every entry is the average value from three placement runs.

4.2 Terminal Propagation

Terminal propagation is the essential technique to the success of min-cut type placement. It leads to better bisection results for placement compared to the *pure* partitioning. This is because terminal propagation uses geometrical information of external terminals.

However, in our placement flow that combines partitioning and simulated annealing, terminal propagation is not as important as it is in min-cut placement flow. The reason is the following. In pure partitioning, although wrong decisions for cells are likely to be made without consideration of external terminals, these mistakes can be fixed in the later annealing stages. In addition, ignoring external connections may entail partitioner to focus on internal connections, leading to a better partitioning solution.

We conduct the following experiments to study terminal propagation in both our flow and min-cut placement flow. Table 2 lists the final wirelength comparison for different terminal propagation usages. Experiments show that in our placement flow, it is better not to start terminal propagation too early or too late. Some medium levels are more reasonable points to start using terminal propagation.

4.3 From Bins to Rows

An inevitable problem for the bin based approach is the difference between the number of rows in the design and the number of rows in the bin grids[5]. Because of this limitation, previous experiments of bin based approach were performed by the aid of detailed placer (e.g., [15]), or on the benchmarks that have 64 or 128 rows (e.g., [14]). We devise a

[5]Unbalanced partitioning (used in min-cut placement) does not apply here, because bin annealing requires that all bins have roughly the same size.

simple-but-effective adjustment step and put it before the cell annealing process. The basic idea is to merge all the cells within the same column in the bin grids, and then evenly divide them into rows. Fig. 3 explains the algorithm.

Input: Placement with overlapped cells at $m \times n$ bins, and number of rows r
Output: Placement at $m \times r$ bins
for ith column in bin grids, $1 \leq i \leq m$ **do**
  Sort all the cells within this column and put them into an array $col()$
  $curr\_width \leftarrow 0$
  for each cell $c \leftarrow col(j)$, $1 \leq j \leq sizeof(col)$ **do**
    $cell\_width \leftarrow$ the width of cell $col(j)$
    $y(j) \leftarrow curr\_width + cell\_width/2$
    $curr\_width \leftarrow curr\_width + cell\_width$
  end for
  $total\_width \leftarrow$ total cell width for cells in $col()$
  for each cell $c \leftarrow col(j)$, $1 \leq j \leq sizeof(col)$ **do**
    $bin\_x \leftarrow$ the original bin x-coordinate for cell $col(j)$
    $bin\_y \leftarrow \lfloor y(j)/(total\_width/r) \rfloor$
    Put cell $col(j)$ into bin (bin_x, bin_y) at new bin grids
  end for
end for

Fig. 3: Algorithm to adjust bins into row structure

This adjustment step does not consider connections between cells, rendering quality loss of wirelength. In experiments we observed about 3% worse wirelength after this adjustment step[6]. A flow-based algorithm could be used for this specific problem and lead to better solution. However, minor loss of quality in this step is acceptable since the following cell annealing will cover the loss.

The adjustment step will make all rows equally long if we can put fractional cells into bins. However, due to the variety of cell widths, the formed rows have different lengths. In experiments we observed $\pm 10\%$ discrepancy for row lengths. Section 4.5 will discuss issues on row balance control.

4.4 Cell Annealing with Bin Structure

We propose a new approach at the cell annealing stage. Specifically, we allow cell moves between bins while changing the centers of bins. Fig. 4 explains the difference between this method and previous approaches. In the figure, (a) and (b) were used in [19]; (c) was used in [11, 14]; (d) is the new method in our approach.

In (a), cell clusters, not cells are swapped during simulated annealing. The freedom of cells are confined by clusters. In (b), cells are next to each other. Moving a cell will change all the locations of cells on the right. Although the authors in [19] employed wirelength estimation technique, this flat annealing is still the most time consuming part. (c) was used in [11, 14]. Its drawback is that all the bins have the same width while every bin has different total cell width. This problem becomes serious especially when the average number of cells per bin is small, or the cell widths vary considerably. In this case the improved wirelength does not correlate to the true wirelength after spreading cells. (d) is new method in our approach. The idea is to keep cells overlapped at bin centers for speeding up cost evaluation process. However, the bin widths are not fixed. The center of a bin will be updated periodically in the simulated annealing[7], according to the summation of bin widths for all the bins on the left. As the temperature becomes lower, less moves are accepted, thus the changes of bin widths become smaller. This procedure of minimizing wirelength will converge at the end. The overlapped

[6]The comparison is made when the number of rows in the bin grids does not change after adjusting. Otherwise the wirelengths are not comparable.

[7]For example, before the temperature change.

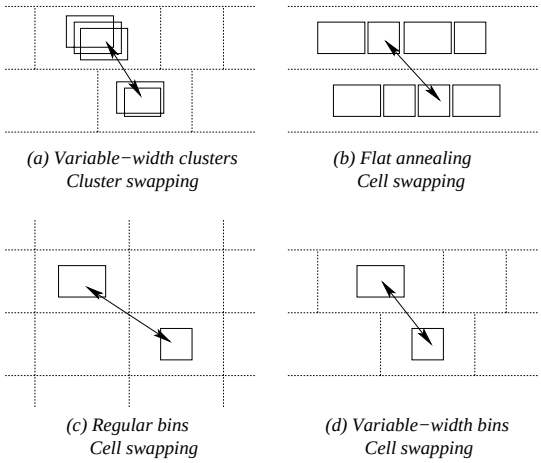

Fig. 4: Comparison between different move types in simulated annealing. (a),(b),(c) are previously used methods and (d) is our new approach.

placement obtained by this method correlates well to the placement after spreading out cells, providing a good initial point for detailed placement.

It should be noted that the method of variable-width bins could be extended to low-density standard-cell designs. White space, or feedthrough area can be assigned into bins and the simulated annealing approach can still be applied. The only difference is that the bins are wider now — it contains not only the cells but also the white space. Detailed placement process needs to be modified accordingly.

4.5 Balance Control

Control of the maximum row length is a very important topic for designs with high row utilization. A gradual budget assignment approach was proposed in [16] on this problem. In our placement flow, the row unbalance comes from the inexact bisections and bin annealing. It is well-known that low tolerances of partitioning result in suboptimal objectives. Moreover, due to the accumulation of the unbalance for a series of partitionings, it is extremely hard to control the row balance by lowering the tolerance in partitioning. Similarly, in the bin annealing stage, banning the cluster moves that violate row balance substantially confines the freedom of clusters and results in loss of placement quality.

The bin adjustment method in Section 4.3 partially helps reducing the row unbalance, yet it cannot eliminate the unbalance. The author in [9] uses a network flow based algorithm to solve the balancing problem. We simplified the flow model in which cells can only be moved between adjacent rows and implemented the similar approach.

The cell annealing stage provides a good opportunity to control the maximum row length. There are two ways to achieve this objective: penalizing the overflowed rows, or disallowing moves that violates row balance. The former needs fine tuning of simulated annealing for appropriate coefficients, while the latter is relatively easy. According to our simplicity principle, we adopt the second approach in our work.

Another detail for the implementation of cell annealing is the choice of move types. We allow both cell swapping and cell shifting, i.e., moving a cell from one bin to another. Experiments show that introducing cell shifting not only improves wirelength results, but also greatly helps the balance control. Moreover, our experience indicates that the flow-based row adjustment method is unnecessary in our flow — cell based annealing solves the balance problem well.

In our experiments, we observed that balance control for very tight design (e.g., 0.01% white space) is very difficult and usually leads to significant loss of quality. Considering that this very tight design is less relevant with current fixed-die issue, we do not further discuss balance control problem for very tight designs.

stage	test circuit 1		test circuit 2	
	random spreading	optimal spreading	random spreading	optimal spreading
after cell annealing	4.50		4.06	
after spreading	4.68	4.55	5.20	4.79
after local impr.	4.52	4.52	4.46	4.41
Impr. at last step	3.4%	0.7%	14.2%	7.9%

Table 3: Comparison of final placement wirelengths using random spreading or optimal spreading. Although optimal spreading gives better wirelength at this step, the final wirelength after local improvement step is similar to that of random spreading.

4.6 Spreading Cells

At the beginning of the final placement stage, we face the problem of spreading the cells within the bins. The authors in [30] use the optimal placer for small placement instances. We integrated the same branch-and-bound algorithm to spread cells in bins with less than 8 cells. However, we found that this step is unnecessary in our placement flow, as the later local improvement covers the difference between an optimal spread and a random spread.

Table 3 shows that the gain from optimal spreading cells is shadowed by the later local improvement step. This is not the first time we have met the situation: an optimization at a given step may not be necessary due to the following optimizations. We hope that the experience obtained from experiments will be helpful for understanding placement problem in a global view.

4.7 Resource Aware Placement

In the over-the-cell routing context, each routing layer has different rules on wire width and wire spacing. Also, a portion (or all) of layer metal-1 usually is occupied by the standard-cells. Thus the routing resources for horizontal wires and vertical wires are indeed not equal. Placement algorithms should take this difference into account to improve routability. However, previous placement work has not mentioned about this issue.

In our benchmarks which use 4-layer or 6-layer routing, the horizontal routing resources are less than vertical ones. Correspondingly, horizontal wirelength should have higher weights compared to vertical wirelength during the placement. We conduct experiments to evaluate the correlation between routability and wire direction.

One way to shorten horizontal wirelength is adding a weight for horizontal length when calculating bounding box of a net. This introduces floating point operations in the bounding box computation, which is the innermost operation of the simulated annealing. Another method is to increase the width of the core area while keeping the height unchanged during the bin annealing stage. This way the horizontal wirelength will be less than that of the original approach as they have higher cost. We expect better routability although the total wirelength will likely increase.

In min-cut type placement, however, the weighting method does not apply. We can use the *cut sequence* method proposed in [8] to affect the horizontal/vertical wirelengths. An observation is: for a partitioning based placement that alternatively uses horizontal and vertical cut, if we use horizontal first cut, the final horizontal wirelength will be shorter than that of using vertical first cut.

Table 4 shows the experimental results on dealing horizontal/vertical wirelengths separately. We run the industrial tool and our placer to get different half-perimeter wirelengths. For each placement run, horizontal, vertical and total wirelengths are reported. Routing results are also reported, including number of violations, routed wirelength and routing time. The table shows that for our placement flow, the horizontal/vertical wirelengths are not significantlly affected by the first cut. However, the *xfactor*, which indicates the weight of the horizontal wires, does affect the lengths of different directions. Also the routability (measured by routing time) becomes better as the horizontal wirelength decreases.

Placer and Options		Placement			Routing		
		total WL	*horizontal WL*	*vertical WL*	*vios*	*WL*	*time*
Industrial	*default*	6.97	2.96	4.01	0	9.34	49
Ours	horizontal first	5.69	2.98	2.71	0	8.09	64
	vertical first	5.59	2.98	2.62	0	7.92	60
	xfactor = 1.1	5.64	2.64	3.01	0	7.58	31
	xfactor = 1.3	5.74	2.88	2.86	0	8.02	42
	xfactor = 1.5	5.76	2.52	3.24	0	7.72	47
	xfactor = 1.7	6.14	2.55	3.59	0	8.24	29

Table 4: Observation on the relationship between routability and horizontal/vertical wirelengths for benchmark *ibm01*. Wirelength is in 10^5 microns. Routing time is in minutes. For each placement run, horizontal and vertical wirelengths are reported as well as total wirelength. We control the first cut as horizontal one or vertical one, and report the result for each run. *xfactor* means that we give higher weights for horizontal wires during placement, resulting in a relatively shorter horizontal wirelength.

The relationship between routability and different wire direction should be studied further. We do not report the results of our placer using xfactor in Section 5 because (a) usually it does not make much difference on routability in our experiments (only routing time changes) and (b) we want to understand the correlation between routability and pure wirelength minimization.

5. Experimental Results

Our placement tool has been implemented in C. We tested the tool on a series of selected benchmarks. These benchmarks originate from the designs used in [22] and [14], and are carefully created as described in Section 2.2. The circuit features are listed in Table 5.

circuits	*cells*	*rows*	*tracks*	*white space*	*row util.*	*core util.*	*routing layers*
matrix	3,083	56	0	1.06%	98.94%	98.94%	4
VP2	8,714	100	0	1.80%	98.20%	98.20%	4
32-MAC	8,902	84	4	1.36%	98.64%	68.29%	4
64-MAC	25,616	143	6	1.64%	98.36%	59.07%	4
ibm01	12,028	106	3	0.68%	99.32%	74.49%	4
ibm02	19,062	146	0	0.67%	99.33%	99.33%	6
ibm03	21,879	149	0	0.32%	99.66%	99.66%	6
ibm04	26,332	174	0	0.61%	99.39%	99.39%	6
ibm07	44,811	222	0	0.92%	99.08%	99.08%	6
ibm08	50,672	240	0	0.71%	99.29%	99.29%	6
ibm09	51,382	235	0	0.91%	99.09%	99.09%	6
ibm10	66,762	307	0	1.00%	99.00%	99.00%	6

Table 5: Tested circuit statistics, including number of cells, number of nets, number of rows, number of routing tracks between rows, row utilization, core utilization, core are and number of routing layers. Core area is in 10^6 square microns.

We compared our placement tool with a well-known industrial placer, Cadence QPlace (Silicon Ensemble, Version 5.3), and a state-of-the-art academic placer, Capo (September 2001 version).[8] All three placers read LEF/DEF files and output placement results in DEF format. We then use Cadence WarpRouter to read the placement outputs and do global and final routing. We consider the routing result *success* (no violation), *finished* (with a small number of violations) or *failure* (too many violations or out of time). In the case of successful and finished routing, the number of violations, routed wirelength and number of vias are reported.

[8]We do not compare the work in [14] (Dragon2000) because: (a) Dragon2000 can not read LEF/DEF files, and (b) Dragon2000 has the limitation on the number of rows.

Table 6 shows the comparison. All the reported wirelengths are in meters and the runtimes are in minutes on Sun Ultra10 workstation with 400MHz CPU. Due to the immense runtime, we only run each placement once.

Compared to QPlace, our placer produces placements with similar or better routability and significantly better layout quality for all tested circuits. For 2 out of total 12 circuits, our placer produces routable placement (without routing violation) while QPlace fails to do so. For 2 circuits both placers fail. For rest of the circuits, our placer's result corresponds to shorter half-perimeter wirelength, shorter routed wirelength and less vias. For some benchmarks, the routing times for the placements produced by our placer are much shorter than those from the QPlace. (*64-MAC, ibm02 and ibm07*). Compared to Capo, our placer shows significantly better ability to produce routable placement.

The wirelength and via improvements by our placement tool are summarized in Table 7. Compared to QPlace, on average we reduce the routed wirelength by 15.3% and vias by 9.1%. The half-perimeter wirelengths produced by our placer are on average 13.2% shorter than those by QPlace. Compared to Capo, on average we reduce the routed wirelength by 18.1% and vias by 8.2%. The half-perimeter wirelengths produced by our placer are on average 14.5% shorter than those by Capo.

Our placement tool usually takes 5 - 10 times longer runtime than QPlace. It is still reasonable. We have tested our placer using the biggest circuit which has 220,000 cells. It takes about eight hours to place it on a PC with 733MHz CPU[9]. We expect that our placer can be speed up by a factor 2 or 3, using approaches better exploiting the bin based data structure. At this moment we focus on quality rather than speed, as a placement with good quality often saves a great amount of time in routing.

Circuits	Placer	Placement		Routing				
		WL	*time**	*result*	*vios*	*WL*	*vias*	*time*
matrix	QPlace	0.10	0.5	*Success*	0	0.12	21556	1
	Capo	0.10	0.4	*Success*	0	0.13	21669	1
	Ours	0.09	2.2	*Success*	0	0.11	21127	1
VP2	QPlace	0.38	1.2	*Success*	0	0.49	69913	15
	Capo	0.36	1.1	*Success*	0	0.51	69257	5
	Ours	0.33	9.1	*Success*	0	0.42	66185	39
32-MAC	QPlace	0.51	1.4	*Success*	0	0.69	95087	12
	Capo	0.56	1.3	*Success*	0	0.74	81382	20
	Ours	0.46	10.9	*Success*	0	0.64	82440	17
64-MAC	QPlace	2.07	5	*Finished*	462	3.40	326344	439
	Capo	2.08	5	*Success*	0	2.88	264713	18
	Ours	1.93	24	*Success*	0	2.60	252092	17
ibm01	QPlace	0.70	3	*Success*	0	0.93	133383	49
	Capo	0.63	2	*Failure*	-	-	-	-
	Ours	0.57	33	*Success*	0	0.81	126861	64
ibm02	QPlace	1.62	7	*Success*	0	2.41	315699	103
	Capo	1.52	4	*Finished*	1	2.35	316261	111
	Ours	1.44	65	*Success*	0	2.11	293244	50
ibm03	QPlace	1.66	6	*Success*	0	2.06	253085	19
	Capo	1.80	4	*Success*	0	2.44	282647	44
	Ours	1.38	40	*Success*	0	1.66	234388	17
ibm04	QPlace	1.92	6	*Finished*	322	2.27	313036	179
	Capo	2.23	6	*Finished*	343	2.66	342134	266
	Ours	1.68	63	*Finished*	343	1.91	279865	142
ibm07	QPlace	3.66	11	*Finished*	50	5.10	621024	414
	Capo	3.90	11	*Failure*	-	-	-	-
	Ours	3.21	76	*Success*	0	4.07	542015	92
ibm08	QPlace	3.98	16	*Finished*	37	5.29	788828	970
	Capo	3.92	12	*Failure*	-	-	-	-
	Ours	3.29	197	*Finished*	20	4.31	712121	316
ibm09	QPlace	3.55	13	*Success*	0	4.23	590681	42
	Capo	3.49	12	*Success*	0	4.10	595388	47
	Ours	2.93	145	*Success*	0	3.42	546233	35
ibm10	QPlace	6.28	22	*Success*	0	7.54	906297	103
	Capo	6.69	17	*Success*	0	8.26	974678	203
	Ours	5.48	276	*Success*	0	6.62	852972	93

Table 6: Comparison between Cadence QPlace, Capo and our placer. Half-perimeter wirelength in placement and routed wirelength are in meters. Runtime is in minutes. A routing is considered failure if the number of violation is greater than 0. (*) Note that the runtimes of QPlace and our placer are on Sun Ultra10 workstation with 400MHz CPU, while the runtimes of Capo are on Intel/PC with 733MHz CPU.

6. Conclusion

The main idea of this paper is a simple-but-good-enough placer for wirelength minimization. We have shown that minimizing wirelength is still the important topic for routability, even in modern fixed-die context. We hope that the simplicity of the placer can help future studies on more complex issues in placement process, such as meeting timing constraints.

We have shown the superior routability of the placements produced by our placer. Compared with the industrial tool and the state-of-the-art academic tool, our placer produces placements with equal or better routability, and much better layout quality after routing. Our future work includes fix-die placement method for designs with general white space, e.g., 50-80%. We believe congestion optimization, as well as wirelength, is the essential step to achieve routability for general fix-die placement.

Circuit	Impr. over QPlace			Impr. over Capo		
	placement wirelength	routed wirelength	vias	placement wirelength	routed wirelength	vias
matrix	10.0%	8.3%	2.0%	10.0%	15.4%	2.5%
VP2	13.2%	14.3%	5.3%	8.3%	17.6%	4.4%
32-MAC	9.8%	7.2%	13.3%	17.9%	13.5%	-1.3%
64-MAC	6.8%	23.5%	22.8%	7.2%	9.7%	4.8%
ibm01	18.6%	12.9%	4.9%	9.5%	-	-
ibm02	11.1%	12.5%	7.1%	5.3%	10.2%	7.3%
ibm03	16.8%	19.4%	7.4%	23.3%	31.2%	17.1%
ibm04	12.5%	15.9%	10.6%	24.7%	28.2%	18.2%
ibm07	12.3%	20.2%	12.7%	17.7%	-	-
ibm08	17.3%	18.5%	9.7%	16.1%	-	-
ibm09	17.5%	19.1%	7.5%	16.0%	16.6%	8.3%
ibm10	12.7%	12.2%	5.9%	18.1%	19.9%	12.5%
average	13.2%	15.3%	9.1%	14.5%	18.1%	8.2%

Table 7: Improvement by our placer compared to QPlace and Capo on half-perimeter wirelength, routed wirelength and number of vias. "-" indicates that the comparison can not be made because of the routing failure.

7. References

[1] M. A. Breuer. "A Class of Min-cut Placement Algorithms". In *Design Automation Conference*, pages 284–290. IEEE/ACM, 1977.

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

[3] C. Sechen and A. Sangiovanni-Vincentelli. "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package". In *Design Automation Conference*, pages 432–439. IEEE/ACM, 1986.

[4] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization". *IEEE Transactions on Computer Aided Design*, 10(3):365–365, 1991.

[5] G. Sigl, K. Doll, and F. M. Johannes. "Analytical Placement: A Linear or a Quadratic Objective Function". In *Design Automation Conference*, pages 427–432. IEEE/ACM, 1991.

[6] S. Goto and E. S. Kuh. "An Approach to the Two-Dimensional Placement Problem in Circuit Layout". *IEEE Transactions on Circuits and Systems*, 25(3):208–214, 1978.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov. "Can Recursive Bisection Alone Produce Routable Placements?". In *Design Automation Conference*, pages 477–482. IEEE/ACM, June 2000.

[9]The placer in [9] spends six hours for a circuit with 200,000 cells.

[8] M. C. Yildiz and P. H. Madden. “Improved Cut Sequences for Partitioning Based Placement”. In *Design Automation Conference*, pages 776–779. IEEE/ACM, 2001.

[9] Jens Vygen. “Algorithms for Large-Scale Flat Placement”. In *Design Automation Conference*, pages 746–751. IEEE/ACM, 1997.

[10] D. Huang and A. B. Kahng. “Partitioning-based Standard-cell Global Placement with an Exact Objective”. In *International Symposium on Physical Design*, pages 18–25. ACM, April 1997.

[11] M. Sarrafzadeh and M. Wang. “NRG: Global and Detailed Placement”. In *International Conference on Computer-Aided Design*. IEEE, November 1997.

[12] H. Eisenmann and F. M. Johannes. “Generic Global Placement and Floorplanning”. In *Design Automation Conference*, pages 269–274. IEEE/ACM, 1998.

[13] X. Yang, M. Wang, K. Eguro, and M. Sarrafzadeh. “A Snap-On Placement Tool”. In *International Symposium on Physical Design*, pages 153–158. ACM, April 2000.

[14] M. Wang, X. Yang, and M. Sarrafzadeh. “Dragon2000: Fast Standard-cell Placement for Large Circuits”. In *International Conference on Computer-Aided Design*, pages 260–263. IEEE, 2000.

[15] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl. “Multilevel Optimization for Large-Scale Circuit Placement”. In *International Conference on Computer-Aided Design*, pages 171–176. IEEE, 2000.

[16] K. Zhong and S. Dutt. “Effective Partition-Driven Placement with Simultaneous Level Processing and Global Net Views”. In *International Conference on Computer-Aided Design*, pages 171–176. IEEE, 2000.

[17] S. Hur and J. Lillis. “Mongrel: Hybrid Techniques for Standard Cell Placement”. In *International Conference on Computer-Aided Design*, pages 165–170. IEEE, 2000.

[18] O. Faroe, D. Pisinger, and M. Zachariasen. “Local Search for Final Placement in VLSI Design”. In *International Conference on Computer-Aided Design*, pages 565–572. IEEE, 2001.

[19] W. Swartz and C. Sechen. “Timing Driven Placement for Large Standard Cell Circuits”. In *Design Automation Conference*, pages 211–215. IEEE/ACM, 1995.

[20] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez. “Unification of Budgeting and Placement”. In *Design Automation Conference*, pages 758–761. IEEE/ACM, 1997.

[21] S. L. Ou and M. Pedram. “Timing-driven Placement Based on Partitioning with Dynamic Cut-net Control”. In *Design Automation Conference*, pages 472–476. IEEE/ACM, June 2000.

[22] Y. C. Chou and Y. L. Lin. “A Performance-Driven Standard-Cell Placer Based on a Modified Force-Directed Algorithm”. In *International Symposium on Physical Design*, pages 24–29. ACM, April 2001.

[23] B. Halpin, C. Y. Chen, and N. Sehgal. “Timing Driven Placement using Physical Net Constraints”. In *Design Automation Conference*, pages 780–783. IEEE/ACM, 2001.

[24] P. N. Parakh, R. B. Brown, and K. A. Sakallah. “Congestion Driven Quadratic Placement”. In *Design Automation Conference*, pages 275–278. IEEE/ACM, June 1998.

[25] M. Wang, X. Yang, and M. Sarrafzadeh. “Congestion Minimization During Placement”. *IEEE Transactions on Computer Aided Design*, 19(10):1140–1148, 2000.

[26] X. Yang, R. Kastner, and M. Sarrafzadeh. “Congestion Reduction During Placement Based on Integer Programming”. In *International Conference on Computer-Aided Design*, pages 573–576. IEEE, 2001.

[27] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. “Multilevel Hypergraph Partitioning: Application in VLSI Domain”. In *Design Automation Conference*, pages 526–529. IEEE/ACM, 1997.

[28] C. J. Alpert, J. H. Huang, and A. B. Kahng. “Multilevel Circuit Partitioning”. In *Design Automation Conference*, pages 530–533. IEEE/ACM, 1997.

[29] W. J. Sun and C. Sechen. “Efficient and Effective Placement for Very Large Circuits”. *IEEE Transactions on Computer Aided Design*, 14(3):349–359, March 1995.

[30] A. E. Caldwell, A. B. Kahng, and I. L. Markov. “Optimal Partitioners and End-case Placers for Standard-cell Layout”. *IEEE Transactions on Computer Aided Design*, 19(no.11):1304–1314, Nov 2000.