2022 CAD Contest Problem D Wirelength-Driven Detailed Macro Placement

B08901024 邱啓翰†, B07901190 邱品誠†, B08901013 曾皓晨†
*Department of Electrical Engineering, National Taiwan University
{B08901024, B07901190, B08901013}@ntu.edu.tw
Instructed by Professor Jie-Hong Roland Jiang

***Abstract*—In this article, we adopt a force-directed approach that adjusts the initial positions of the macros to optimize the overall half-perimeter wire length(HPWL) between all macros and standard cells. Given the initial placement of macros and standard cells with the maximum placement constraint, our main algorithm consists of three steps. (1) estimate the position of standard cells by force-directed method to facilitate the second step. (2) determine the position of macros by force-directed method to reduce overall HPWL. (3) consider the placement constraints and the overlapping issues with different macros to obtain the final placement position. We also implement our parser to read input files in Verilog, DEF, and LEF format. We further implement the output part to export the positions of macros after placement. Moreover, an additional C++ program is written to convert the format of the output file into the one in ntuplace3. Compared with the initial placement given by the ICCAD organizers, HPWL results conducted by ntuplace3 show that our method could reduce the overall HPWL without violating the displacement constraint and overlapping constraint in several cases.**

I. Introduction

As technology advances, IC circuits nowadays often contain both standard cells and macros, which are referred to as mixed-size circuits. One of the most popular ways to deal with the mixed-size placement is the three-stage approach ([2]). The three stages are placement prototyping, macro placement, and standard-cell placement respectively. In the given input files, the ICCAD organizers have already done the first stage(placement-prototyping) for us. Our main algorithm would focus on the second stage(macro placement) to adjust the positions of macros given the maximum displacement constraint. After we determine the positions of macros, we would send our results to the ntuplace3 placer([4]) to conduct the third stage(standard-cell placement). Finally, we would use the resulting detailed HPWL given by the ntuplace3 as a standard to determine the feasibility of our algorithm.

A. Previous Work

We had studied several literatures([1], [2]) before we conceived the main algorithm for the macro placement. Most of them let the macros move freely during the placement phase. The integrated spreading algorithm provided in [2] considers both macros and standard cells positions and also introduces a two-step cell-size control technique, new force-modulation technique, and congestion-region-aware macro-shifting technique to cope with large-scale mixed-size circuits. While [1] provides a novel damped-wave framework that incorporates both bottom-up macro clustering and top-down macro declustering techniques to achieve better routability and avoid overflows on large-scale mixed-size circuits. These previous works provided us with some insights into macro placement algorithms. Nevertheless, they did not focus on how to adjust the positions of the macros given the displacement constraint. In addition, they did not illustrate the details of dealing with the possible macro-overlapping issues during the placement. As a result, we finally adopt the force-directed method which we learned in class to deal with the macro placement problem with the maximum displacement constraint.

B. Our Work

Different from what we studied in literature [1] and [2], we present a force-directed algorithm to cope with ICCAD Problem D, namely, a macro placement problem with given constraints. We summarize our major work as follows:

- We construct our parsers to read files in Verilog, DEF, LEF format and write output files about the results of macros' positions.
- We estimate the positions of standard cells after ntuplace3 placer places them by force-directed approach to facilitate the macro placement process.
- We implement a force-directed algorithm to adjust the positions of macros under the displacement constraint and also handle the overlapping macros issue.
- We create another small program to convert our output file into the format which could be accepted by ntuplace3 placer.
- We test our placing result by using ntuplace3 placer to finish the standard-cell placement part and obtain the detailed HPWL about the mixed-size circuits.

The remainder of this article is organized as follows: Section II gives preliminaries about the problem formulation of ICCAD contest Problem D and the data structures we use. Section III proposes our main algorithm for macro placement. Section IV

shows the resulting HPWL calculated by ntuplace3. Furthermore, we discuss about the experimental results in V. Finally, we conclude our work in Section VI.

II. Preliminaries

A. Problem Formulation

Given the initial position of standard cells and macros and the circuit netlist, the macro placement problem in ICCAD contest problem D requires us to find the minimum total half-perimeter wire length(HPWL) by adjusting the positions and orientations of the macros under the following constraints:

- The orientation is restricted to North, FlipNorth, South, and FlipSouth.
- The maximum displacement distance is provided in a text file such that the Manhattan distance between an initial and final position of every macro is limited.
- The macro could not overlap with each other.
- The final position of the macro could not exceed the boundary of the die.
- The running time of the program could not exceed 20 minutes.

B. Data Structure

We also construct several Class in C++ and make use of some data structures to organize the circuit netlist. Three of the major class are described below:

- A Component class which stores the name, position, orientation, whether it is movable or not, which component type it is belongs to, and the connection of each pin. This class is used to store the information of both standard cells and macros described in the DEF file and save the connection in the Verilog file.
- A Component_type class which stores the name, the size, and the pin location. This class is used to store the properties of each standard cell and macro block described in the LEF file.
- A Pin class which stores the name, and the position. This class is used to store the location of primary input and primary output pins described in the DEF file. Besides, it can also be used in the member of the Component_type class to describe the pin location.

In addition, considering the 20 minutes time constraint, we adopt Unordered_map(dictionary) as our main data structure in the program since the searching time for an Unordered_map is theoretically O(1). If the data need to be traversed once during the algorithm, we choose the data structure to be Vector because of the maintainability.

III. Algorithm

We adopt the **force-directed approach** to move each macro to the zero-force position. For each movable macro, consider all the cells connected to it, and calculate its zero-force position. The detailed calculating method is illustrated below.

A. Standard Cell Placement

Given a standard cell, we find the connection of each pin and adopt the forced-directed approach described in the next section to estimate the approximate position of the standard cell after being placed by ntuplace3. Unlike the macro placement technique, we do not check for the overlapping of each standard cell to save our computation time.

B. Forced-directed Approach

We calculate the zero-force position based on the result in the *standard cell placement (III-A)*: for each pin on a given macro, find all pins of the other cells that connect to this macro through this pin, and sum over all the x and y coordinates, respectively, of those connected pins. Repeat this for all of the pins on this macro.

We reference to the formula used in [3]:

$$\hat{x_i} = \frac{\sum_j w_{ij} a_j}{\sum_j w_{ij}}$$

, where $\hat{x_i}$ represents the zero-force position, w_{ij} represents the weight of a wire, and a_j represents the position of a connected pin on another cell.

However, since each pin of a given macro is located differently, we have to take the position of each pin into account. We then use the following formula to calculate the ideal position which we would like to place the macro to:

$$\hat{x_i} = \frac{\sum_j w_{ij}(a_j - b_j)}{\sum_j w_{ij}}$$

, where $\hat{x_i}$ represents the zero-force position, w_{ij} represents the weight of a wire, a_j represents the position of a connected pin on another cell, and b_j represents the position of the pin that connects to a_j.

C. Orientation of Components

We notice the fact that all the pin positions in .lef is based on the consumption that the whole component is in the North direction, nevertheless the actual pin positions are determined by the orientation of the macro, which is defined in .def. Thus, whenever we want to access an exact position of a pin, we should consider both the relative position of the pin, the exact position of the macro, and the orientation of the macro. Our algorithm to handle the orientation problem is as follows:

1) North:
   both x and y stay the same.
2) FlipNorth:
   relative position of x coordinate become macro_width - original_relative_position.
3) South:
   relative position of both x and y coordinates become macro_width - original_relative_position.
4) FlipSouth:
   relative position of y coordinate become macro_width - original_relative_position.

D. Maximum Displacement Constraint

Due to the maximum displacement constraint of each macro, we often cannot place a macro to the ideal position calculated in *forced-directed approach (III-B)*. This is because the ideal position is too far from the initial position of the macro. Therefore, a method to handle the problem is developed as follows. According to the ideal position previously calculated,

1) First, a) if a macro should move longer along x-axis than y-axis, then move it along x-axis until "its distance to the ideal position along x-axis is the same as that along y-axis" or "the macro movement reaches maximum displacement"; on the contrary, b) if a macro should move longer along y-axis than x-axis, then move it along y-axis.
2) Second, if the macro movement doesn't reach maximum displacement, then move it toward the ideal position until it reaches the constraint.

E. Boundary of the Chip and Macro Overlapping Handling

Due to the fact that the optimal solution we calculate in the previous step might overlap another macro, and that the ntuplace3 would not function correctly if the overlapping macro are specified as FIXED in the placement file, we need to adopt an effective algorithm to prevent all macros from overlapping. The following procedure will repeat several times until the operating macro finishes tuning. First, we check if the operating macro is out of the bound of the whole chip. If it happens, we reassign the optimal position to a legal place near the boundary. Second, we traverse all macros excluding itself and put them into handle_overlap function. Once an overlapping happens in an iteration, we need to check all the macros again for fear that the new optimal position might overlap with macros that are modified previously.

The check_overlap function will examine two given rectangles, each includes x-coordinates, y-coordinates, width and height. The algorithm would first calculate the left, right, top and bottom boundaries of each rectangle. After that, it would check if one of the left or right boundary of first rectangle is between the left bound and right bound of the second rectangle, then it would check if one of the top or bottom bound of first rectangle is between the top bound and bottom bound of the second rectangle. Once the two condition is fulfilled, we can say that the two rectangle overlaps. In the same logic above, we could ensure overlap issues happen if one of the left or right bound of the second rectangle is between the left bound and right bound of the first rectangle and one of the top or bottom bound of the second rectangle is between the top bound and bottom bound of the first rectangle simultaneously.

The handle_overlap function will first call check_overlap function to make sure whether the given two macro overlaps. If the two overlaps each other, there are two solutions:

1) For fear that some macros which we will move later do not have the legitimate position, before moving macro, we check whether any overlap occurs. If that happens, we quit that movement.

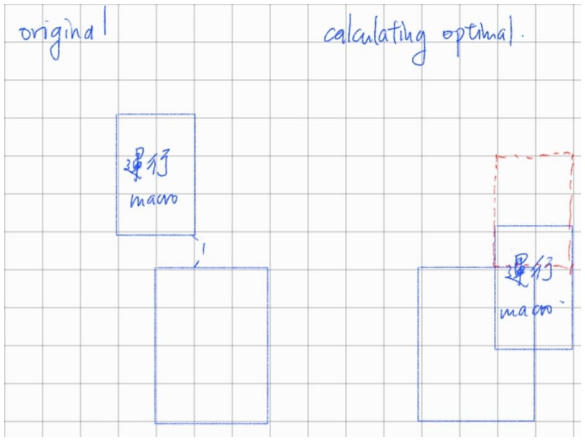

Fig. 1.
1. Because initial position overlaps in x-coordinate, choose y-direction to tune.
2. The legal maximun distance in y diretion is 1.
3. Set the new moving distance of y-direction to 1 and remain the moving distance of x-direction. The new optimal position is the red box.

2) The previous method may not help us converge to an optimal solution, so the second solution is that when overlapping issues happen, try to move the operating macro to the best position without overlapping other macro. We first determine the direction of the overlapping. If the initial position of the operating macro overlaps in x-coordinate, we will tune it in y-direction. Otherwise, we will tune it in x-direction. If both coordinates of the position doesn't overlaps, we will calculate whether the difference between the maximum which x can legally move and the move_distance_x, or the difference between the maximum which y can legally move and the move_distance_y is bigger. We will tune the macro in the bigger direction of the two. Also, for fear that it never converges, we restrict that the moving distance of the two coordinate can only decrease or maintain the same. Thus, our method is to change the macro's moving distance to maximum distance along x or y-axis previously calculated on the previous chosen direction while remaining the moving distance in the other direction. Noting that our calculation of max distance must be smaller than current optimal moving distance. The details are in Fig. 1.

IV. Experimental Results

Case#	Original Detailed HPWL(G)	Ours(G)
01 (1^{st})	223.5	232.4
01 (2^{nd})	223.5	224.9
02	319.8	287.6
03 (1^{st})	348.5	340.9
03 (2^{nd})	348.5	344.5

```
Circuit: case01.aux
  Global HPWL= 232223638568  Time:   2122 sec (35.4 min)
   Legal HPWL= 251154756689  Time:      1 sec (0.0 min)
  Detail HPWL= 232419133953  Time:    284 sec (4.7 min)
 ============================================================
         HPWL= 232419127113  Time:   2436 sec (40.6 min)
```

Fig. 2. our case01 result using 1^{st} boundary method and without standard cell placement

```
Circuit: case01.aux
  Global HPWL= 231889402858  Time:   4247 sec (70.8 min)
   Legal HPWL= 243042780897  Time:      1 sec (0.0 min)
  Detail HPWL= 224938332097  Time:    547 sec (9.1 min)
 ============================================================
         HPWL= 224938325573  Time:   4843 sec (80.7 min)
before log result
```

Fig. 3. our case01 result using 2^{nd} boundary method and with standard cell placement

```
Circuit: case02.aux
  Global HPWL= 288126045902  Time:   3673 sec (61.2 min)
   Legal HPWL= 305512883922  Time:      1 sec (0.0 min)
  Detail HPWL= 287587425025  Time:    337 sec (5.6 min)
 ============================================================
         HPWL= 287587421039  Time:   4056 sec (67.6 min)
```

Fig. 4. our case02 result using 1^{st} boundary method and without standard cell placement

```
Circuit: case03.aux
  Global HPWL= 351359311741  Time:   2239 sec (37.3 min)
   Legal HPWL= 358436051210  Time:      1 sec (0.0 min)
  Detail HPWL= 340877554689  Time:    438 sec (7.3 min)
 ============================================================
         HPWL= 340877549933  Time:   2720 sec (45.3 min)
before log result
```

Fig. 5. our case03 result using 1^{st} boundary method and without standard cell placement

```
Circuit: case03.aux
  Global HPWL= 349199115881  Time:   3726 sec (62.1 min)
   Legal HPWL= 371237573761  Time:      1 sec (0.0 min)
  Detail HPWL= 344479212247  Time:    387 sec (6.5 min)
 ============================================================
         HPWL= 344479205464  Time:   4153 sec (69.2 min)
```

Fig. 6. our case03 result using 2^{nd} boundary method and with standard cell placement

V. Discussion

Though the second solution utilize a lot more techniques, including better boundary algorithm and our own standard cell placement, we still get a worse result comparing to our first approach. We think that there are several reasons to this phenomenon:

1) The new boundary method may not improve a lot on these issues.
2) Our initial placement is different from the final placement by ntuplace3.

Due to 20 min time constraint, our initial placement may be too simple, contributing little benefit to the following macro displacement.

We also record the time when we run our program. Usually, the portion of time spent between reading input, conducting standard cell placement, and determining final macro position is 6% vs 91% vs 3%. We spent most of the time in estimating the position of standard cells because the number of standard cells are much larger than that of macros. In the future, we might focus on improving the estimation of the standard cells with a view to finding a quicker and more reliable way.

VI. Conclusions

We have designed an algorithm to deal with the macro placement problem in ICCAD Contest Problem D. In the experimental results, we could see that our algorithm is capable of reducing detailed HPWL in case2 and case3. Although the detailed HPWL increases in case1, the global HPWL of case1 is smaller than that of original placement. As a result, we infers that the force-directed approach might still effect the HPWL globally.

VII. Work Distribution Table

Student	Distribution	Contribution
邱品誠	input output parser, main algorithm	33%
曾皓晨	main algorithm	33%
邱啓翰	main algorithm	33%

References

[1] Chin-Hao Chang, Yao-Wen Chang, and Tung-Chieh Chen. “A novel damped-wave framework for macro placement”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2017, pp. 504–511.

[2] Szu-To Chen, Yao-Wen Chang, and Tung-Chieh Chen. “An integrated-spreading-based macro-refining algorithm for large-scale mixed-size circuit designs”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2017, pp. 496–503.

[3] N. Quinn and M. Breuer. “A forced directed component placement procedure for printed circuit boards”. In: *IEEE Transactions on Circuits and Systems* 26.6 (1979), pp. 377–388. DOI: 10.1109/TCS.1979.1084652.

[4] The-OpenROAD-Project. *Replace/ntuplace at standalone · the-OpenROAD-Project/Replace*. URL: https://github.com/The-OpenROAD-Project/RePlAce/tree/standalone/ntuplace.