2012

Minimum Cost Flow Based Fast Placement Algorithm

Enlai Xu
Iowa State University

Minimum cost flow based fast placement algorithm

by

Enlai Xu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:

Chris Chu, Major Professor

Zhao Zhang

Doug Jacobson

Iowa State University

Ames, Iowa

2012

DEDICATION

To my parents, my major professor Chris Chu and my cousin Zion Shen.

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

ACKNOWLEDGMENTS viii

ABSTRACT ix

CHAPTER 1. Introduction 1

1.1 Introduction to Placement 1

1.2 Previous Work 4

1.3 Our Work 7

CHAPTER 2. Overview of MCF Fast Placement Algorithm 9

CHAPTER 3. MCF Based Technique for Rough Legalization 14

3.1 Problem Formulation 14

3.1.1 Formulate Rough Legalization problem to MCF Problem 14

3.1.2 Reduce the Runtime of the Algorithm 17

3.2 Cell Moving Algorithm 19

3.3 Handling Macros 27

3.3.1 Fix Movable Macros 27

3.3.2 Use MCF Based Approach 27

CHAPTER 4. Refinement 29

4.1 Iterative Local Refinement 29

4.2 Slice Based Refinement 31

CHAPTER 5. Experimental Results 38

CHAPTER 6. Conclusion 41

LIST OF TABLES

5.1 HPWL(*10e6) comparison on the ISPD-2005 placement contest benchmark. 38

5.2 HPWL(*10e6) comparison on the ISPD-2006 placement contest benchmark. 39

5.3 Runtime comparison on the ISPD-2005 placement contest benchmark. 39

LIST OF FIGURES

1.1 Typical VLSI Design Flow 2
1.2 Typical Placement Flow 3
1.3 HPWL of a Net 5
1.4 Placement Category 7
2.1 Two Level of Hierarchy MCF Based Algorithm 11
3.1 Rough Legalization 15
3.2 Formulate Rough Legalization Problem to Minimum Cost Flow Problem 16
3.3 Flow Value Interpretation 18
3.4 Limit the Target Range of Target Bins to Reduce Runtime . . . 19
3.5 Eliminate Bins with no Free Space 20
3.6 Move Cell to Target Bin 24
3.7 Fix Movable Macros 27
3.8 Fix Movable Macros 28
4.1 ILR 8 Target Positions 30
4.2 Reordering Technique 31
4.3 Row Refinement 32
4.4 Column Refinement 32
4.5 Optimal Region 33
4.6 Cell Distribution before Refinement 34

4.7 Cell Distribution after Refinement 34
4.8 Cell Distribution after Quadratic Program 35
4.9 Cell Distribution after MCF based Rough Legalization 36
4.10 Cell Distribution after Slice based Refinement 37

ACKNOWLEDGMENTS

I would like to use this opportunity to thank those who helped me on my research and this thesis.

First of all, thank Dr. Chris Chu to give me this great opportunity to study my Master Degree in Iowa State University. And thank him to bring me such an advanced research topic in VLSI CAD area. During these years, his very kind guidance shows me the way how to setup a research topic, how to model a research problem, and how to solve a problem with a creative method. He gave me so many precious suggestions that would be very helpful for my future work.

Furthermore, I would also like to express my thanks to my other committee members, Dr. Zhao Zhang and Dr. Gary Tuttle, for their insightful suggestions. I learned much useful knowledge for my research topic from them including Dr. Zhao Zhang's course, CPRE581 Computer System Architecture, and Dr. Doug Jacobson 's CprE 530 Information Warfare. It is a great honor to study from and working along with them.

Finally, I would like to dedicate this thesis to my parents and my cousin who gave me great support during my master degree.

ABSTRACT

Placement is a very important and critical procedure during VLSI design. In this thesis, we propose a new force-directed quadratic placement algorithm called Minimum Cost Flow Based Fast Placement Algorithm(MCFPlace) for large-scale circuits. We mainly have three contributions:

(1) In this thesis, we propose a novel flow for global placement. The main idea is to generate a relatively good placement at very early stage during the iterations of quadratic program and addition of move force. After quadratic program, we apply a rough legalization technique to spread out the cells and some refinement techniques to generate a placement of good quality. We then use this placement as target positions and add move force to more effectively guide the movement of cells.

(2) In order to generate target positions of cells with very good quality, we first perform a rough legalization. we propose a new Minimum Cost Flow (MCF) based approach for rough legalization which spread the cell evenly over the whole placement region at a global level. The approach not only spreads cells at a global level, but also takes the wirelength into consideration.

(3) Furthermore, we incorporate some refinement techniques after Minimum Cost Flow based approach. We propose a novel slice based refinement technique and incorporate the iterative local refinement(ILR) technique to further improve the quality of placement of target positions of cells. By doing this, we can have a more accurate move force to more effectively guide the cell movements at the very early stage which will have a great impact on the final quality of placement.

Our placer is 1% better than RQL[16], 1.35% better than SimPL[3], 3.5% better than

mPL6[22], 4% better than FastPlace3[23] in terms of wirelength for ISPD05 benchmark suites. On runtime our placer is 1.5 times faster than RQL and 4.4 times faster than mPL6. Though we are 1.3 percent worse than the current best placer Maple[4] in terms of wirelength, our algorithm is 4 times faster than Maple.

CHAPTER 1. Introduction

1.1 Introduction to Placement

Placement is a very fundamental and critical procedure in VLSI design. As Fig. 1.1. shows, after floorplanning, placement will be applied to determine the locations of the circuit modules in the placement region. The quality of a placement will affect the circuit greatly on its performance, routability, power consumption and distribution of heat. In placement, total wirelength is the most commonly used objective, since wirelength is the key factor in determining the performance of a circuit. It determines the delay of interconnect wires. As feature size in advanced VLSI technology continues to reduce, interconnect delay can consume as much as 75% of clock cycle in advanced design. Besides, minimizing the total wirelength will also indirectly optimize several other objectives like routability, power consumption and so on.

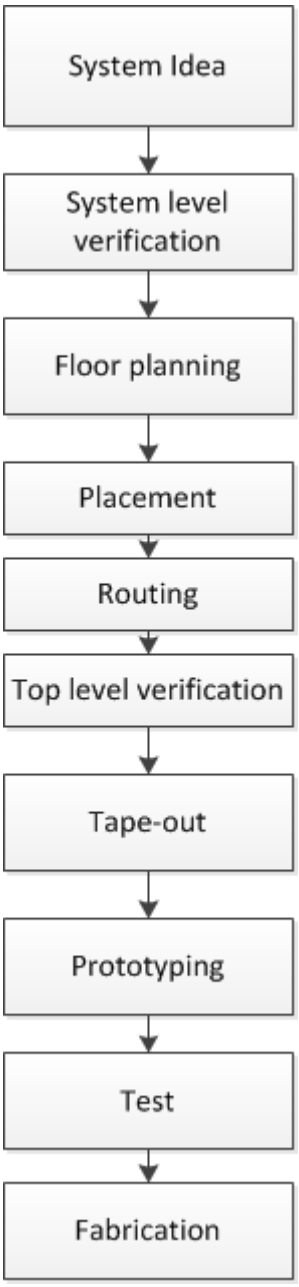

Figure 1.1 Typical VLSI Design Flow

Placement problem is a complex problem. People usually do placement in several manageable steps to make the placement easier. One common flow has three major steps: Global Placement, Legalization and Detailed Placement, as Fig.1.2 show:

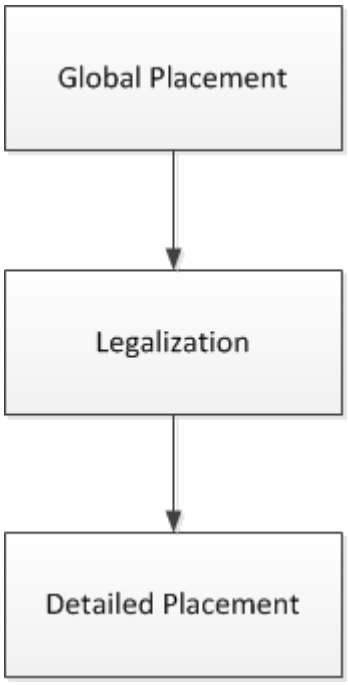

Figure 1.2 Typical Placement Flow

1. Global placement aims at generating a rough placement solution that may violate some placement constraints (e.g., there maybe overlaps among modules) while maintaining a global view of the whole netlist.

2. Legalization makes the rough solution from global placement legal (i.e., no placement constraint violation) by moving modules around locally.

3. Detailed placement further improves the legalized placement solution in an iterative manner by rearranging a small group of modules in a local region while keeping all other modules fixed.

The global placement step is the most important one out of the three. It has the most impact on placement solution quality and runtime, and has been the focus of most prior research works. After global placement, the placement solution is almost completely determined. In legalization and detailed placement, only local changes in module locations will be made. The main emphasis of this thesis is the global placement step.

1.2 Previous Work

For the past few decades, people have done a lot of research on placement problem. In wirelength driven placement, the main objective is to minimize the total weighted wirelength. People have achieved tremendous progress in how to spread cells while minimizing the wirelength.

State-of-the-art placers can be classified into three main categories: stochastic approaches, partitioning approaches and analytical approaches.

(1) Placers based on stochastic approaches often utilize simulated annealing. This optimization method theoretically can achieve the global optimum but suffers from long CPU run times. Two representative of stochastic placers are Timberwolf [18] and Dragon[21].

(2) Partitioning approach is to recursively partition the circuit and the placement region. PROUD [15] partitions the circuit based on the locations of the modules as determined by quadratic placement. The min-cut placers Capo [9] and FengShui[1] partition the circuit based on a certain cost function such as number of wires crossing a boundary of adjacent partitions.

(3) Nowadays, analytical approach is more popular for large-scale circuits, because it is faster in runtime and has better quality than other two approaches. The main idea of analytical placers is to express the objective function and constraints of placement as analytical formulas. In other words, the placement problem is formulated as a mathematical program. Depending on the kind of objective function, analytical placers can be subdivided into two categories: Nonlinear-Optimization-based placers and Quadratic placers.

In a Nonlinear-Optimization-based placer, the objective function is nonlinear, e.g., a log-sum-exponential function [25], which is minimized by nonlinear optimization techniques like conjugate-gradient optimization. APlace [11], mPL[20], and NTUPlace [19]

are all typical examples of nonlinear-optimization-based placers.

In a Quadratic placer, its objective function is quadratic and can therefore be minimized efficiently by solving a system of linear equations. Following placers are all Quadratic placer: Gordian [10], Kraftwerk [5], FAR [7]/mFAR [8] , FastPlace [24], BonnPlace [2] hATP [6], FDP [12], RQL [16], Kraftwerk2 [17] and SimPL[3].

Most of state-of-the-art analytical placers are Quadratic placers, since they can achieve good quality placement at low CPU time. But there are two main issues that quadratic placer should handle. The first issue is to achieve minimum wirelength using a certain net model. The second issue is to spread module to remove the overlap among them. For the first issue, Kraftwerk2 [17] proposed a linear BoudingBox net model which is an exact and deterministic presentation of the half-perimeter wirelength(HPWL) in a quadratic objective function. The HPWL of a net is equal to half of the perimeter of the smallest bounding retangles that encloses all the pins of the net as Fig.1.3 shows.

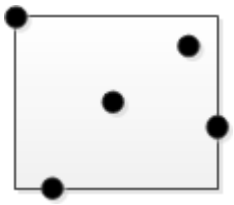

Figure 1.3 HPWL of a Net

In this thesis we mainly discuss the second problem: to spread cells over the placement region. People usually have two ways to spread cells in a quadratic placer. One is to add center of mass constraints like Gordian[10], BonnPlace [2] and hATP [6]. To refine the center-of-mass constraints, these quadratic placers often partition the placement area recursively and assign modules to the placement partitions.

The other is to use force-directed method to spread cells like Kraftwerk[5], FAR[7], mFAR[8], FastPlace[24], FDP[12] and SimPL[3]. Before Kraftwerk2[24], different approach try to implement the additional force needed for this category of quadratic placement. For example, Kraftwerk [5] utilizes the module density to determine a constant

additional force, which moves the modules from high to low-density regions. FDP utilizes a similar approach as [5].FAR calculates the additional force like [5] but models it by fixed points. mFAR [8]uses two different fixed points to express the additional force. The perturbing fixed points reduce module overlap and are calculated heuristically by a local bin utilization. The controlling fixed points achieve the force equilibrium and are determined also by heuristics. FastPlace[24] uses cell shifting technique for the additional force as mFAR[8].

Kraftwerk2[24] proposed a more robust approach by separate the additional force into two fundamental components: move force and hold force. In this approach, the placer will generate the target position of each cell. And then move force will be added according to target position of each cell. They use a generic demand-and-supply formulation of the placement and the potential formulation to calculate the nonheuristic move force.

The force-directed quadratic placers are very popular because of the following two reasons: First it is much faster than nonlinear-optimization-based placers like mPL6. Second, it can also achieve placement with very good quality.

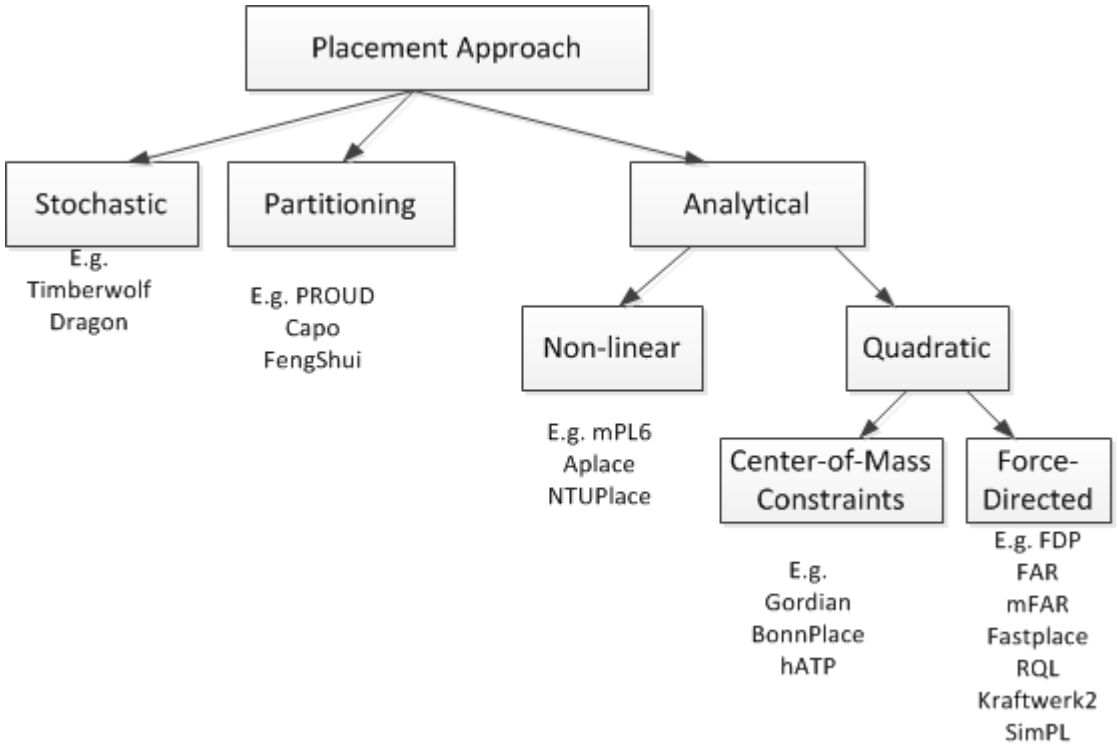

Figure 1.4 Placement Category

1.3 Our Work

In this thesis, we propose a new algorithm for global placement. The core of our new algorithm is to generate a relative good placement by rough legalization, new slice based refinement technique and iterative local refinement during the iteration stage of quadratic program and addition of move force. We have tree major contributions:

(1) In a quadratic placer, after quadratic program, moving force will be added to guide the movement of cells. We propose a new algorithm that generates very accurate move force for cells after quadratic program. After quadratic program, we will first use Minimum Cost Flow based approach to perform a rough legalization to spread out cells over whole placement region, then we use our new slice based refinement technique and iterative local refinement technique to improve the wirelength. We then use this placement as target positions of cells and add move force to more effectively guide the movement of cells which will have a great impact on the final quality of placement.

(2) In order to generate target positions of cells with very good quality, we first perform a rough legalization. we propose a new Minimum Cost Flow (MCF) based approach for rough legalization which spread the cell evenly over the whole placement region at a global level. The approach not only spreads cells at a global level, but also takes the wirelength into consideration.

(3) Furthermore, we incorporate some refinement techniques after MCF based approach. We propose a novel slice based refinement technique and incorporate iterative local refinement(ILR) technique to further improve the quality of placement of target positions of cells. By doing this, we can have a more correct move force to more effectively guide the placement at the very early stage which will have a great impact on the final quality of placement.

Our placer is 1% better than RQL[16], 1.35% better than SimPL[3], 3.5% better than mPL6[22], 4% better than FastPlace3[23] in terms of wirelength ISPD05 benchmark suites. On runtime our placer is 1.5 times faster than RQL[16] and 4.4 times faster than mPL6[22]. Though we are 1.3 percent worse than the current best-quality placer MAPLE[4] in terms of wirelength, our placer is 4 times faster than MAPLE[4].

CHAPTER 2. Overview of MCF Fast Placement Algorithm

In a typical iterative force directed placement algorithm, after solving the quadratic program, move force and hold force will be added for next iteration of quadratic program. The way to add move force will make a big influence on placement quality. In order to add a very accurate move force to more effectively guide the movement of cells, it is very necessary to apply some refinement techniques at early stage.

In this thesis, we propose a new algorithm called MCF Fast Placement Algorthm(MCFPlace) for global placement by incorporating MCF based rough legalization technique, ILR refinement technique and slice based refinement technique in generating the move force. The flow of our new global placement algorithm is as follows:

New Placement Algorithm: MCFPlace

1 Repeat

2 quadratic program

3 MCF based approach for rough legalization

4 Refinement

5 Add move force to spread cells

6 Until all modules are roughly spread and has no significant improvement.

In this flow of our new algorithm, after quadratic program, we will first use MCF based approach for rough legalization to spread out cells. We will present this technique in details in Chapter 3. This technique will spread out the cells over the whole placement

region evenly. The MCF based approach can roughly legalize the placement and also take the wirelength into consideration.

Though the MCF based approach takes the wirelength into consideration, it still introduces some errors because of the limitation of problem formulation. To generate a placement with better quality, we employ some refinement techniques to further improve the wirelength. Therefore after MCF based approach for rough legalization, slice based refinement technique or Iterative Local Refinement will be applied to further reduce the wirelength. Chapter 4 will discuss those two refinement techniques in more details. Slice based refinement technique is a more global reordering technique than Local Reordering in FastPlace. Our technique can fix both horizontal and vertical errors. And ILR is a technique proposed in FastPlace, which is more accurate. But compared with slice based refinement, it is much slower. Therefore, for the first several iterations we will use slice based technique and after that ILR will be applied to do the refinement work.

In practice, we implemented our new algorithm in a hierarchy manner. We employ a two-level clustering scheme which is proposed in FastPlace3. We call this new algorithm hMCF Fast Placement Algorithm.

As Fig.2.1 show, hMCF Fast Placement Algorithm(hMCFPlace) employ a two-level clustering scheme. In the first level of clustering, it will create fine-grain clusters of about 2-3 objects per cluster.This clustering is solely based on the connectivity information between the objects in the original flat netlist. Since this clustering is performed before any placement, it will restrict it to fine-grain clustering to minimize any loss in placement quality due to incorrect clustering.

After the first level of clustering, the algorithm will perform a fast, initial placement of the fine-grain clusters. The purpose of this step is to get some placement information for the next clustering level. Since each cluster in the first level has only around 2-3 objects, the initial placement of the clusters closely resembles an initial placement of the flat netlist. The algorithm then creates coarse-grain clusters by performing a second level

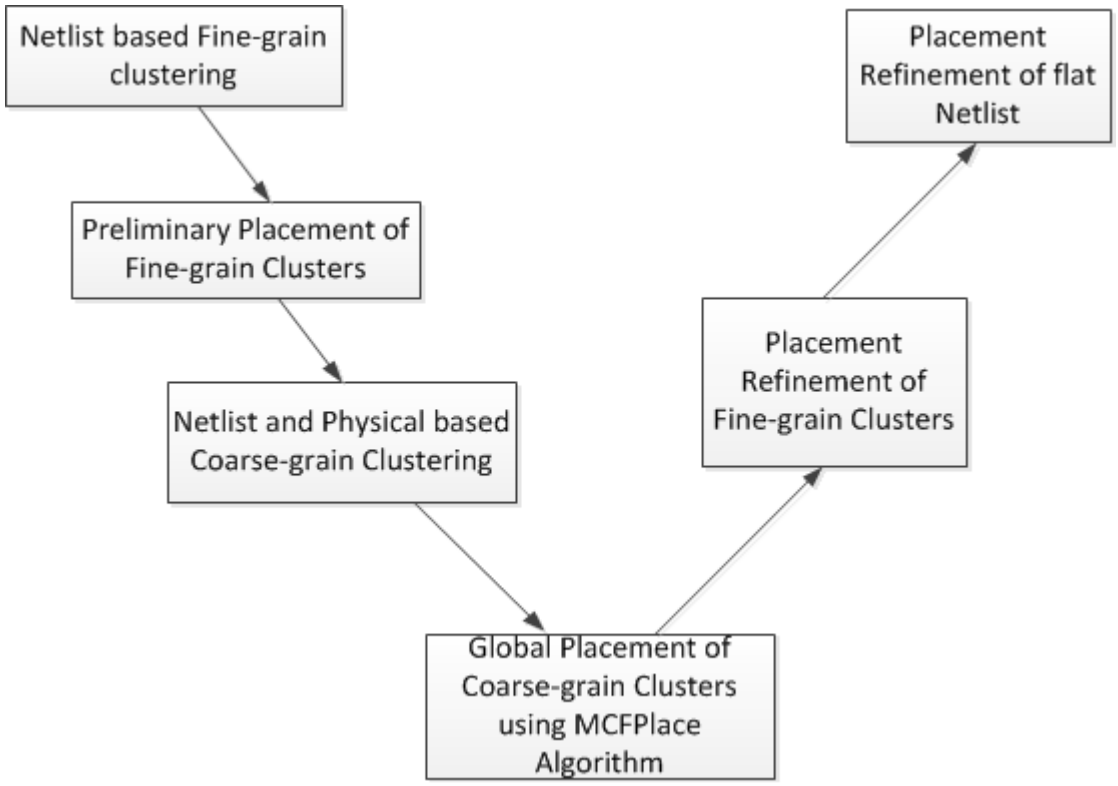

Figure 2.1 Two Level of Hierarchy MCF Based Algorithm

of clustering. In this level, we consider both, the connectivity information between the clusters and their physical locations as obtained from the initial placement. Generating coarse-grain clusters based on actual placement information, is better than generating them by a solely netlist based approach. Also, such an approach would further minimize any loss in (or even improve) the final wirelength.

Base on this clustered netlist, we employ MCFPlace placement algorithm to generate a roughly legalized placement. This step is the core step of our algorithm. Then we will uncluster coarse-grain clusters and perform ILR based refinement on the fine-grain clusters. After that we uncluster fine-grain clusters and perform ILR based refinement on the flat netlist. The whole detailed steps of our hMCFPlace placement algorithm are as follows:

hMCF Fast Placement Algorithm(hMCFPlace)

Stage 1: Initial Placement

1. Construct fine-grain clusters using netlist based clustering
2. Solve initial quadratic program
3. Repeat

a. Perform regular Iterative Local Refinement on fine-grain clusters

4. Until the placement is roughly even

Stage 2: Coarse Global Placement

5. Construct coarse-grain clusters using netlist and physical based clustering
6. Repeat

a. Solve the convex quadratic program

b. MCF based approach rough legalization, refinement and add spreading forces

7. Until the placement have no significant improvement after rough legalization
8. Repeat

a. Perform density-based Iterative Local Refinement on coarse-grain cluster

b. Perform regular Iterative Local Refinement on coarse-grain clusters

c. Perform cell-shifting on coarse-grain clusters

9. Until the placement is quite even

Stage 3: Refinement of fine-grain clusters

10. Un-cluster coarse-grain clusters
11. Perform density-based Iterative Local Refinement on fine-grain clusters
12. Perform regular Iterative Local Refinement on fine-grain clusters

Stage 4: Refinement of flat netlist

13. Un-cluster fine-grain clusters
14. Perform density-based Iterative Local Refinement on flat netlist
15. Perform regular Iterative Local Refinement on flat netlist

Since we apply refinement technique at very early stage and two levels of clustering, our placer can achieve placement of very good quality at low CPU runtime. In our experiment, we do the global placement by hMCFPlace Placement algorithm first, then we apply Fast Detailed Placer[14] to do legalization and detailed placement. The results will be discussed in Chapter 5 in details.

CHAPTER 3. MCF Based Technique for Rough Legalization

After quadratic program, adding accurate move force will make a big difference on the quality of the placement. In Kraftwerk2, they use a generic demand-and-supply formulation to calculate the nonheuristic move force. SimPL will perform a rough legalization based on top-down recursive geometric partitioning and non-linear scaling. The main issue of their approaches is that they can't take wirelength into consideration. Our algorithm not only spread cells more globally but also take wirelength into consideration.

3.1 Problem Formulation

3.1.1 Formulate Rough Legalization problem to MCF Problem

We formulate the spreading problem as a minimum cost network flow problem.

If we divide the whole placement region into N*N bins, each bin has an initial cells. Some bins have more cells and some bins have fewer cells. And if we want to do a rough legalization at bin level, we must move cells from high-density bins to low-density bins. For example in Fig. 3.1, if the total area of cells in bin B0 exceeds the maximum amount bin B0 can accommodate, we must move cells from bin B0 to other bins.

In this problem, there are two objectives we need to achieve. First is to spread cells evenly at bin level. Second is to minimize wirelength as much as possible. To solve this problem, we formulate the problem with Minimum Cost Flow problem with demand and supply as Fig.3.2

In Fig. 3.2, we have two sets of bins to presents the N*N bins of the whole placement

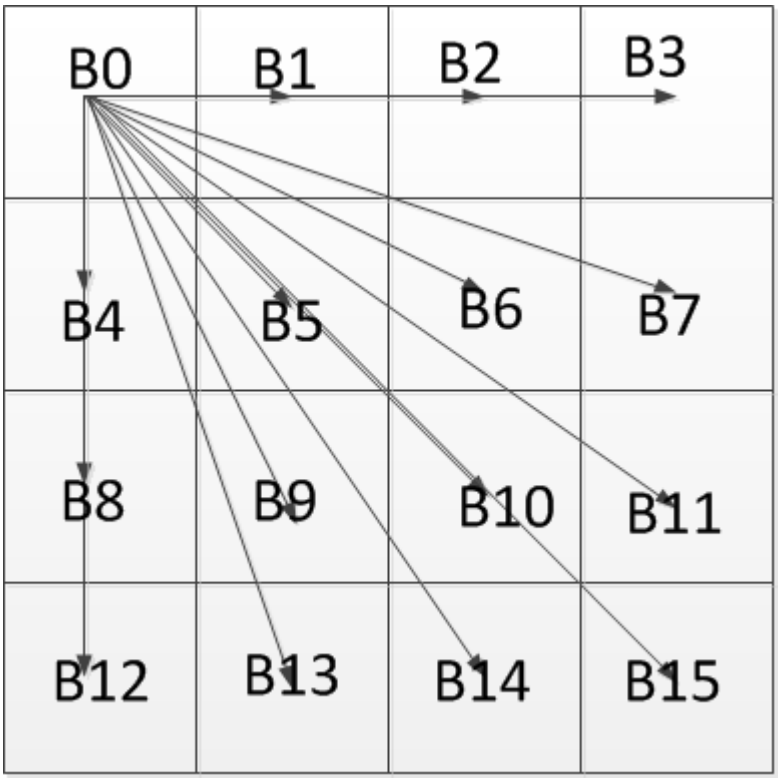

Figure 3.1 Rough Legalization

region.

- We use left column nodes to presents bins that act as source bins in the whole placement region. Every bins can be source bin, no matter there are any cells in that bin or not. Therefore we have N*N nodes in the left column. The supply of these nodes represents the current cell distribution. It is sum of the cell area in that corresponding bin. If there no cell in that corresponding bin, the supply of that node will be zero.

  Supply of bin j=sum of total area of cells in Bin j;

- We use right column nodes to presents bins that act as target bins. Since every bins can be the target bin, we have N*N nodes in the right column to represent them. The supplies of all right column nodes are 0, because they only act as target bins.

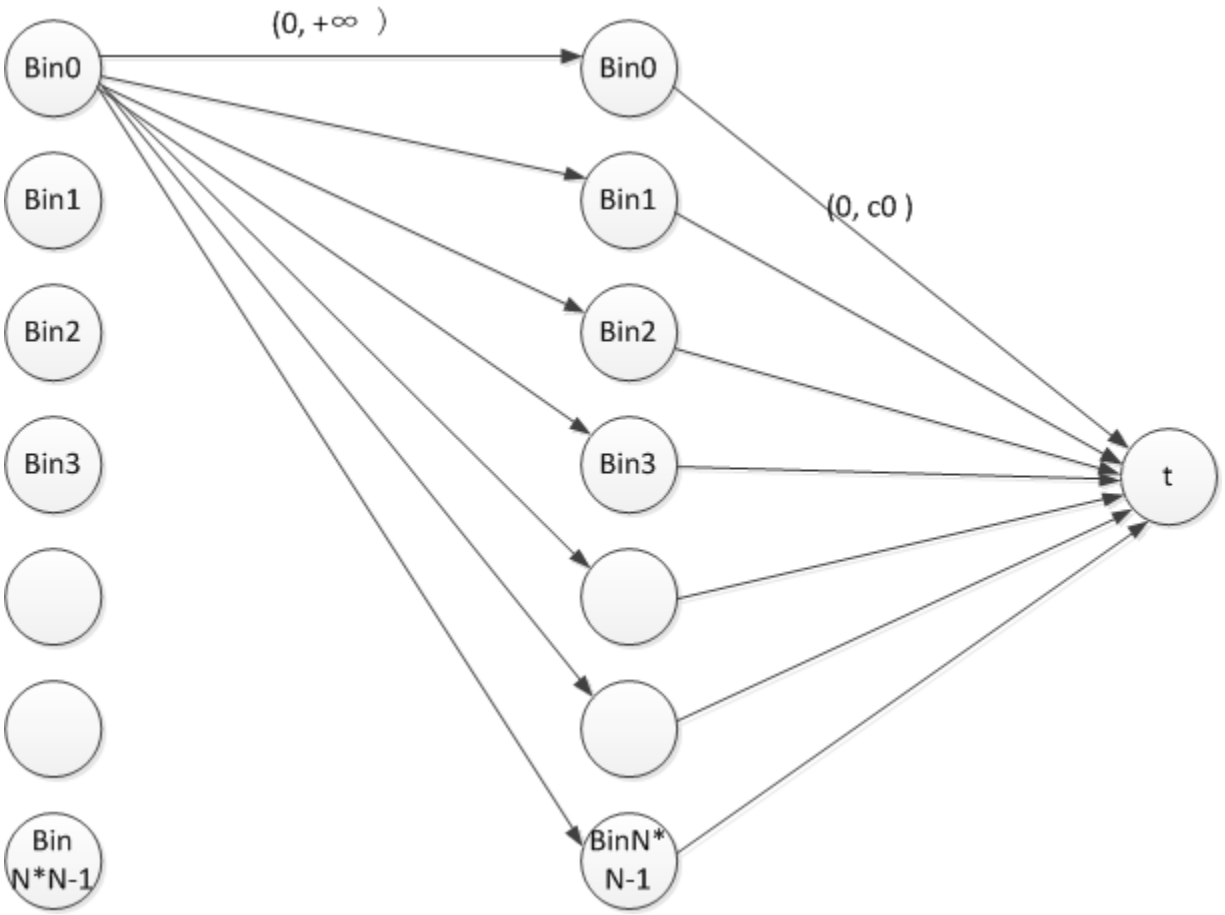

Figure 3.2 Formulate Rough Legalization Problem to Minimum Cost Flow Problem

- The t node will have a demand of total movable cell area in the placement region. The t node will guarantee that all the overflowed bins will move cells to other bins (including themselves).

- We set the capacity of arcs from left column nodes to right column nodes as positive infinity. And set the capacity of arcs from right column node i to node t as(0,ci) . Ci represents the free space of bin i. It will guide the flow and guarantee that there will be no overflow in each bin.

  For standard cells placement, the free space is the area of the bin. In practice, we set the it as D*Bin Area. For fixed macro placement, we will first calculate the utilization of each bin according to the fixed macros. Then Free space=D*(Bin

Area-Utilization). D is the density parameter. We set it as 98% in our algorithm.

- In order to minimize the wirelength increase. We add costs in the arcs from left column nodes to right column nodes. The represents the potential wirelength change of cell moving. It is hard to catch actual wirelength change of cell moving. In this algorithm we set the cost(i,j) as the Euclidean distance of bin i and bin j.

- The bin size is very important in both wirelength and runtime of the algorithm. Setting bin size correctly will make a big difference. Smaller bins size will have a better quality and more even distribution. But too small bins size will increase the runtime of solving MCF problem. We divide the placement region into N*N bins. By experiment, we set N from 15 to 60;

After formulating the problem as above stated, we solve the MCF problem using the free MCF library provided by ZUSE-INSTITUT BERLIN[13]. The library is a simplex implementation which is fast and robust to solve our MCF problem. We store flow value on arc[i][j] form left column nodes to right column nodes in an array flow[i][j], i is the index of left column node and j is the index of right column node. Flow[i][j] means we must move flow[i][j] amount of cells from Bin i to Bin j. Flow[i][j] doesn't represent the number of cells but the total area of cells to move. In most cases, there will be a flow from a bin to itself. In this case, flow[i][i] means there will be flow[i][i] amount of cells remaining in bin i.

For example, in Fig. 3.3, we must move flow[4][1] amount of cells from B4 to B1, flow[4][8] amount of cells from B4 to B8 and flow[4][3] amount of cells from B4 to B3. And flow[4][4] is the total area of cells that will stay in B4.

3.1.2 Reduce the Runtime of the Algorithm

Though the library can solve the problem at a very fast speed. For example, if we employ 50*50 bins, the runtime will be less than 1s. However our problem size is not

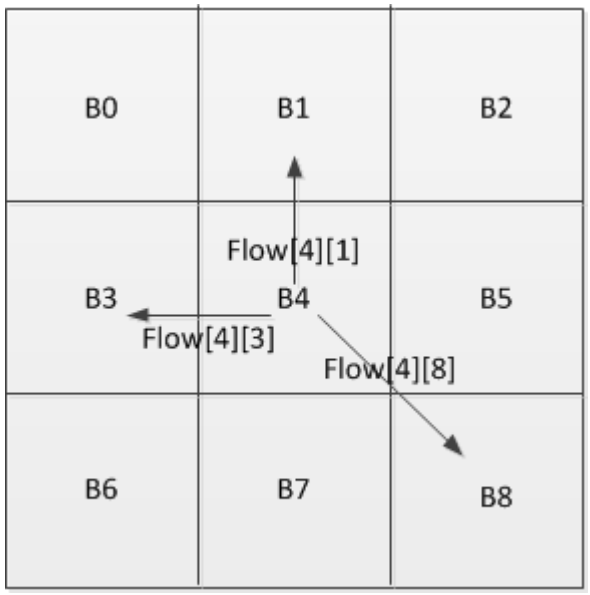

Figure 3.3 Flow Value Interpretation

linear if we increase the bin number. Assume we have N*N number of bins, then we will have N*N*(N*N+1) number of arcs. If we increase parameter N to 2N, the problem will increase by 16 times.

In order to reduce the runtime, we reduce the number of arcs by limiting the target period;

In Fig. 3.4, for a particular bin, we will not allow it to move cells to every other bin. We only allow it to move cells to its adjacent bins within certain range. For example, in Fig.3.4 the range of target bin is 2. This will reduce the number of arcs significantly and speed up the technique.

This may cause infeasibility of the flow solution especially for first iterations when a lot of cells are clustered in the center. Therefore, for the first iterations, we will set the range of target bins big enough to include all the bins, and as the cells being spread out more and more, we will decrease the range of target bins.

Furthermore, some bins which have no free space. We delete their corresponding nodes from the right column. And some bins which have no initial cell distribution. We can also delete their corresponding nodes from left column.

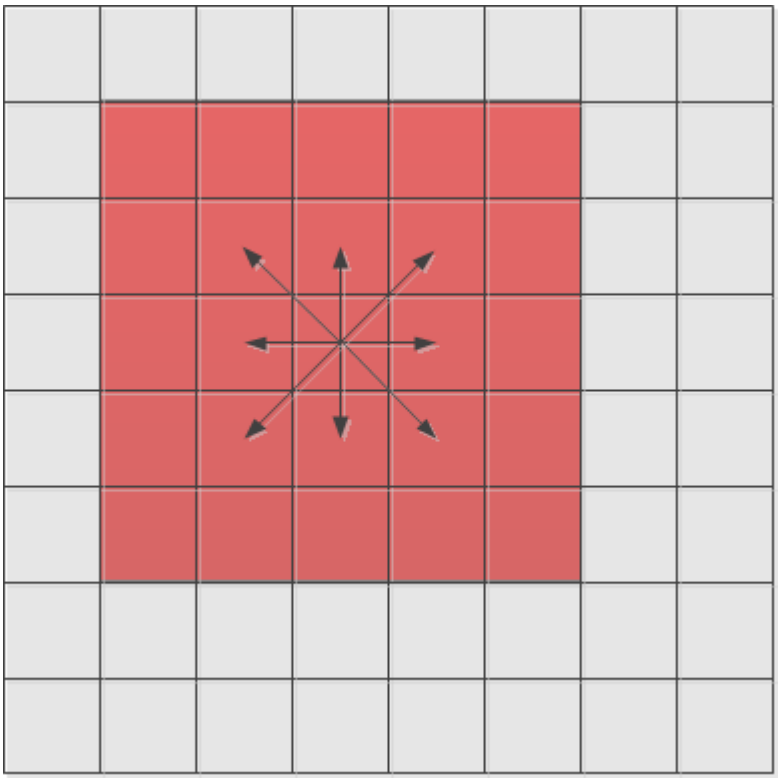

Figure 3.4 Limit the Target Range of Target Bins to Reduce Runtime

For example, in Fig. 3.5 B4 has no free space. We can delete node B4 from right column to reduce runtime.

3.2 Cell Moving Algorithm

According to the flow value from bin to bin, we need to move cells from bin to bin. There two issues in spreading cells from original bins to target bins. First is to satisfy the flow as much as possible. Second is to minimize the wirelength as much as possible.

There are many cells in a bin, we need to select proper cells to move. And for a particular cell, it may has many optional target positions. It is important to choose the proper target position. Our main strategy is as follows: calculating the wirelength cost of moving a cell to every optional target bin. Sort these moving options according to the wirelength cost. Choose from the minimum cost one until the flow is fulfilled.

A very simple implementation of the above strategy is, for each bin, to sort every

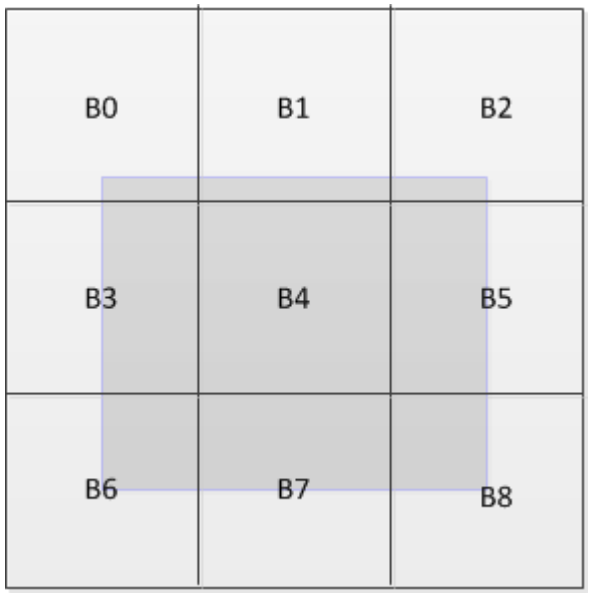

Figure 3.5 Eliminate Bins with no Free Space

option of every cell in that bin once. Then move all the cells in one iteration. This method is fast but not accurate enough. Only the cells in a bin are moved, their position will be updated. This will make cost estimation not accurate enough. Besides, too many cost(j,k) will make sorting cost(j,k) be the bottleneck of runtime of the algorithm.

```
01 For each cell
02    isMoved[i]=0;
03 End for
04 For each bin i (in the topologically sorted order)
05    For each cell j in the bin i
06       For each target bin k
07          Compute the cost(j,k) of moving cell j to bin k;
08       End for
09    End for
10    Sort cost(j,k) in ascending order;
11    For each cost(j,k)
```

```
12      If moved[j]=0 and flow[i][k] bigger than 0 then
13          Move cell j to bin k;
14          isMoved[i]=1;
15          flow[i][k]-=cellArea[j];
16      End if
17    End for
18 End for
```

Another way is to move 10% of cells of each bin every round. By updating those moved cells, the cost estimation will be more accurate. The disadvantage is that it will much slower by repeatedly calculating cell moving cost and sorting.

```
01 For each cell
02    isMoved[i]=0;
03 End for
04 While
05    For each bin i (in the topologically sorted order)
06       For each cell j in the bin i
07          For each target bin k
08             Compute the cost(j,k) of moving cell j to bin k;
09          End for
11       End for
12       Sort cost(j,k) in ascending order;
13       tmpflow=0;
14       For each cost(j,k)
15          If moved[j]=0 and flow[i][k] bigger than 0 then
16             Move cell j to bin k;
```

```
17              isMoved[i]=1;
18              flow[i][k]-=cellArea[j];
19              tmpflow+=cellArea[j];
20              if(tmpflow bigger than F*flow[i][k]) break;
21            End if
22          End for
23      End for
24 End While
```

F is the parameter we use to control the speed and quality of moving. Smaller F will have more runtime but better quality and vice versa. The while Loop will end until all the flows are fulfilled.

According to the above analysis, there are two main factors will affect the quality and runtime of the algorithm. First is the policy on cell position update. Second is the runtime on calculating and sorting cost(j,k); We come up with a faster and more accurate method to select and move cells with more frequent update and less calculation and sorting of cost(j,k);

```
01 For each cell
02    isMoved[i]=0;
03 End for
04 While
05    For each bin i (in the topologically sorted order)
06       For each cell j in the bin i
07          For each target bin k
08             Compute the cost of moving cell j to bin k;
09          Get the minimum costmin(j,k) of the cell j;
```

```
10          End for
11        End for
12        Sort costmin(j,k) in ascending order;
13        tmpflow=0;
14        For each cost(j,k)
15          If moved[j]=0 and flow[i][k] bigger than 0 then
16            Move cell j to bin k;
17            isMoved[i]=1;
18            flow[i][k]-=cellArea[j];
19            tmpflow+=cellArea[j];
20            if(tmpflow bigger than F*flow[i][k]) break;
21          End if
22        End for
23    End for
24 End While
```

We can still use F as the parameter to control the speed and quality.

There are two trivial issues in this algorithm. First issue is the target position of a moved cell. If we want to move a cell to target bin, we can put the cell at the center of target bin or at the relative position of target bin. The former will have cells more clustered within a bin, therefore we put a cell at the relative position in the target bin as Fig.3.6 shows.

The second issue is that after a cell is moved, we will update flow[i][j] by subtract the cell are. Ideally, the total area of moved cells are exactly the same with flow[i][j]. But actually it is impossible to satisfy the flow value exactly. Because the smallest flow unit is 1, but typical area of a standard cell is much bigger. In our algorithm, we handle this issue in this way: continue cell moving until the flow[i][j] is less than zero. And in

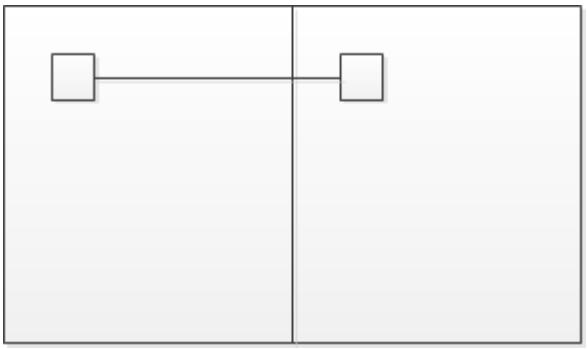

Figure 3.6 Move Cell to Target Bin

reality, there little difference between ideal distribution and actual distribution.

Take adaptec1 benchmark for instance. We divide the placement region into 10*10 bins. The initial cell distribution density is as follows(density of bin i=total area of cells in bin i/area of bin i):

$$\begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.02 & 0.02 & 0.02 & 0.01 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.12 & 0.60 & 0.10 & 0.02 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.02 & 0.02 & 0.76 & 1.43 & 1.57 & 3.45 & 0.04 & 0.02 & 0.00 \\ 0.00 & 0.02 & 0.04 & 1.74 & 5.59 & 0.88 & 3.30 & 0.13 & 0.00 & 0.00 \\ 0.00 & 0.02 & 0.19 & 0.80 & 4.72 & 4.20 & 0.17 & 0.05 & 0.00 & 0.00 \\ 0.00 & 0.03 & 0.18 & 0.29 & 1.21 & 0.43 & 0.04 & 0.11 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.06 & 0.05 & 0.06 & 0.02 & 0.03 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix} \quad (3.1)$$

The free spaces of bins are as follows:

$$\begin{pmatrix} 0.58 & 0.23 & 0.25 & 0.30 & 0.41 & 0.20 & 0.25 & 0.67 & 0.61 & 0.58 \\ 0.95 & 0.55 & 0.22 & 0.27 & 0.46 & 0.16 & 0.22 & 0.66 & 0.95 & 0.95 \\ 0.95 & 0.91 & 0.86 & 0.79 & 0.74 & 0.60 & 0.69 & 0.66 & 0.95 & 0.95 \\ 0.95 & 0.81 & 0.60 & 0.63 & 0.53 & 0.12 & 0.41 & 0.34 & 0.91 & 0.95 \\ 0.90 & 0.82 & 0.71 & 0.74 & 0.63 & 0.41 & 0.57 & 0.51 & 0.83 & 0.95 \\ 0.25 & 0.86 & 0.95 & 0.62 & 0.38 & 0.72 & 0.74 & 0.66 & 0.79 & 0.38 \\ 0.30 & 0.88 & 0.95 & 0.67 & 0.55 & 0.61 & 0.90 & 0.90 & 0.81 & 0.16 \\ 0.30 & 0.72 & 0.79 & 0.65 & 0.02 & 0.11 & 0.11 & 0.15 & 0.18 & 0.43 \\ 0.19 & 0.38 & 0.40 & 0.37 & 0.02 & 0.11 & 0.11 & 0.15 & 0.10 & 0.18 \\ 0.63 & 0.38 & 0.53 & 0.58 & 0.37 & 0.43 & 0.43 & 0.45 & 0.39 & 0.48 \end{pmatrix} \tag{3.2}$$

After solving MCF problem, the ideal cell distribution density should be as follows:

$$\begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.23 & 0.41 & 0.13 & 0.02 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.22 & 0.27 & 0.46 & 0.16 & 0.22 & 0.66 & 0.01 & 0.00 \\ 0.00 & 0.11 & 0.86 & 0.79 & 0.74 & 0.60 & 0.69 & 0.66 & 0.95 & 0.00 \\ 0.00 & 0.81 & 0.60 & 0.63 & 0.53 & 0.12 & 0.41 & 0.34 & 0.91 & 0.05 \\ 0.02 & 0.82 & 0.71 & 0.74 & 0.63 & 0.41 & 0.57 & 0.51 & 0.83 & 0.13 \\ 0.02 & 0.86 & 0.95 & 0.62 & 0.38 & 0.72 & 0.74 & 0.66 & 0.79 & 0.00 \\ 0.00 & 0.88 & 0.95 & 0.67 & 0.55 & 0.61 & 0.90 & 0.90 & 0.81 & 0.00 \\ 0.00 & 0.00 & 0.79 & 0.65 & 0.02 & 0.11 & 0.11 & 0.15 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.07 & 0.37 & 0.02 & 0.11 & 0.11 & 0.05 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.05 & 0.07 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{3.3}$$

After cell movement, the real cell distribution is as follows:

$$\begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.23 & 0.41 & 0.13 & 0.02 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.22 & 0.27 & 0.46 & 0.16 & 0.22 & 0.65 & 0.01 & 0.00 \\ 0.00 & 0.11 & 0.86 & 0.79 & 0.74 & 0.60 & 0.69 & 0.66 & 0.95 & 0.00 \\ 0.00 & 0.81 & 0.60 & 0.63 & 0.53 & 0.12 & 0.41 & 0.34 & 0.91 & 0.05 \\ 0.02 & 0.82 & 0.71 & 0.74 & 0.62 & 0.41 & 0.57 & 0.51 & 0.83 & 0.13 \\ 0.02 & 0.86 & 0.95 & 0.62 & 0.38 & 0.72 & 0.74 & 0.66 & 0.80 & 0.00 \\ 0.00 & 0.88 & 0.95 & 0.67 & 0.55 & 0.61 & 0.90 & 0.91 & 0.81 & 0.00 \\ 0.00 & 0.00 & 0.79 & 0.65 & 0.02 & 0.11 & 0.11 & 0.14 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.07 & 0.36 & 0.02 & 0.11 & 0.11 & 0.05 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.05 & 0.07 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{3.4}$$

Difference of ideal distribution and actual distribution is as follows

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .015 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .015 & 0 & 0 & .001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .07 & 0 & 0 \\ 0 & 0 & 0 & .027 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.5}$$

We can see, our cell moving algorithm can almost achieve the ideal distribution generated by MCF solution. Only 5% of bins have small differences which can be ignorable.

3.3 Handling Macros

Our network flow problems can handle fixed macros and small movable macros very well. But it is very hard to handle movable macros directly in the network flow problem when the area of movable macro is bigger than the area of network flow bin.

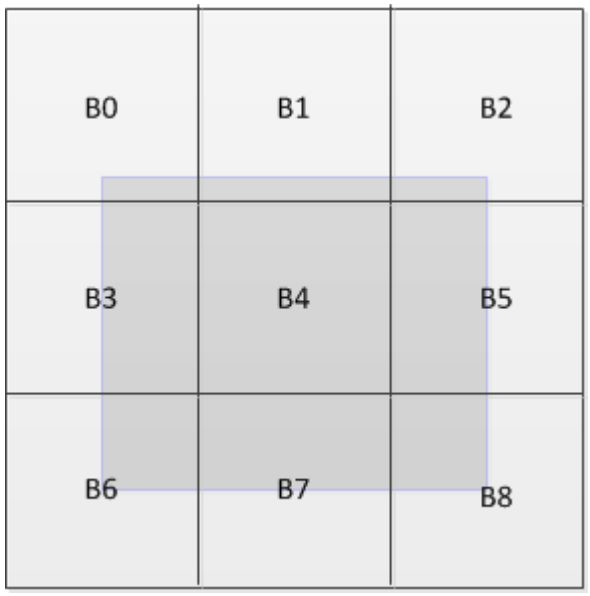

Figure 3.7 Fix Movable Macros

We handle the problem in this way. First fix the movable macros and then formulate MCF network problem based on the fixed macros.

3.3.1 Fix Movable Macros

It is very critical to fix the movable macros at the right position. One heuristic method is to fix the movable macros according to the quadratic program. The quadratic program gives the optimal positions of the movable macros. Then we will fix the movable macros at the positions generated by quadratic program.

3.3.2 Use MCF Based Approach

Once the movable macros are fixed, regarding them as fixed macros. We can easily spreading all the other cells by repeating the above MCF based approach. As Fig. 3.8

show, after fixed the movable macros, the standard cells will be more spread out. If you only regard the movable macros as standard cells, there will be more overlaps between the macros and other cells.

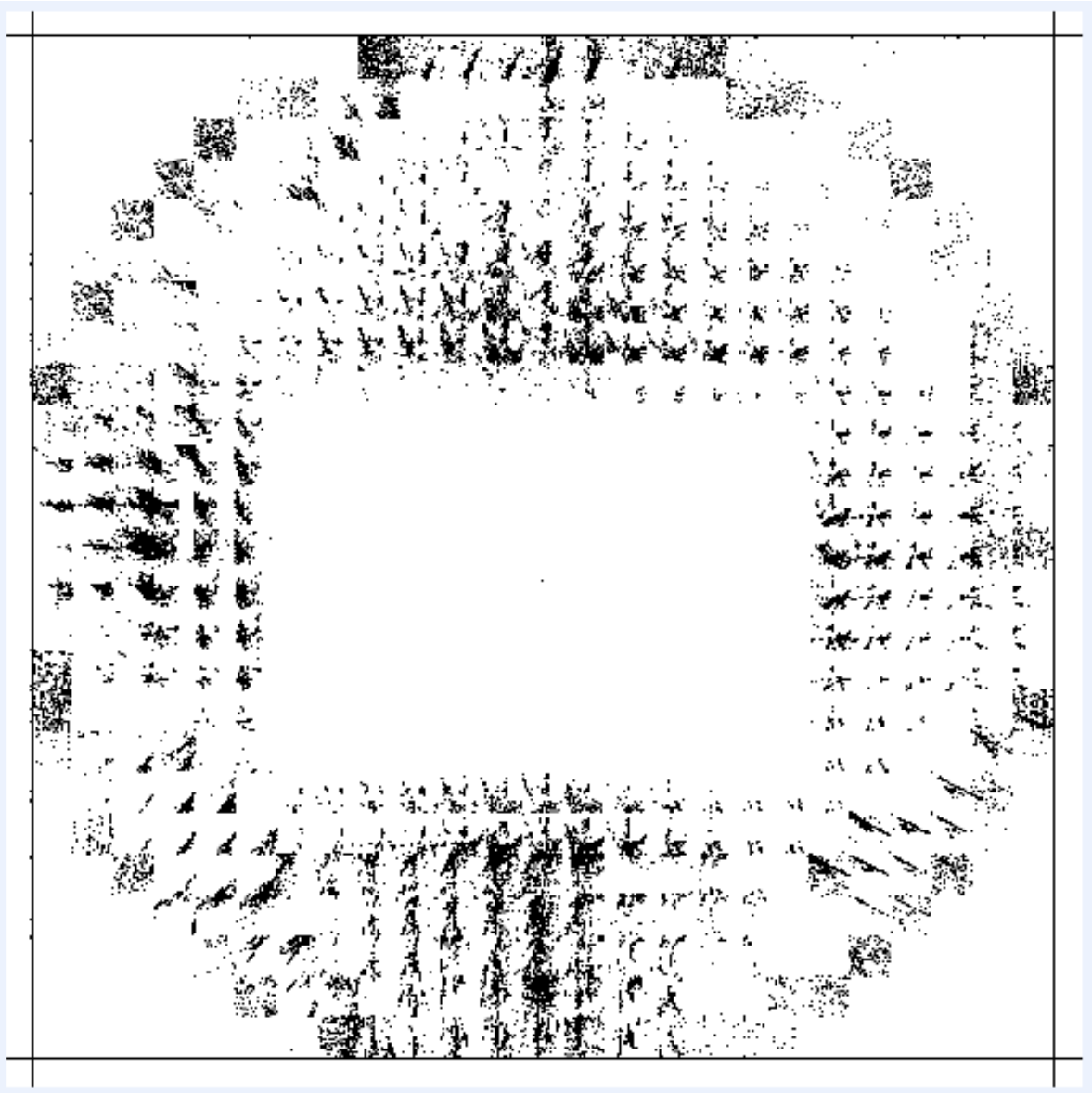

Figure 3.8 Fix Movable Macros

CHAPTER 4. Refinement

Though the MCF based approach takes the wirelength into consideration, it still introduces some errors because of the limitation of problem formulation. To generate a placement with better quality, we employ some refinement techniques to further improve the wirelength. Therefore after MCF based approach for rough legalization, slice based refinement technique or Iterative Local Refinement will be applied to further reduce the wirelength.

4.1 Iterative Local Refinement

Iterative Local Refinement(ILR) is very effective in improving wirelength. It is a technique to reduce the wirelength directly according to HPWL and actual position of a cell. We apply ILR as refinement technique in our new flow to correct mistakes made in MCF based approach.

ILR employ bin structure to estimate the utilization of a placement region and move modules. And For every module it will compute eight scores of that correspond to moving the module to its eight adjacent bins like Fig.4.1. The cell will be moved to the bin that has the biggest score.

The way to calculate the score as follows: it assumes that a cell is moving from its current position in a source bin to the same relative position in the target bin. The score for each move is a weighted sum of two components: the first being the change in the wirelength for the move and the second being a function of the change in the bin

utilization. The wirelength component is computed as the sum of the half-perimeter

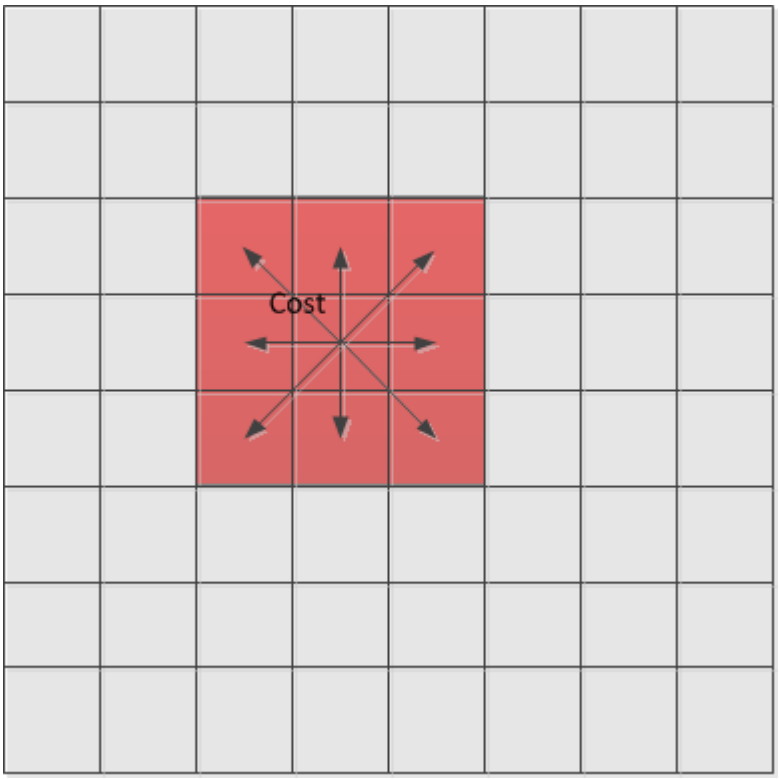

Figure 4.1 ILR 8 Target Positions

of the bounding rectangle of all the nets connected to the module. Since it directly takes the HPWL into account, it is more accurate than the quadratic objective function. For the utilization component to accurately reflect the placement distribution, we define a utilization weight for each bin in the placement region. This weight is a function of the bin utilization and is constantly updated based on the current placement distribution. Hence, a sparse bin will have a low utilization weight so that more modules can be moved out of the bin. As the weights are a function of the bin utilization, they are constantly updated and prevent oscillations in terms of the movement of the modules. If all eight scores are negative, the module will remain in the current bin. Otherwise it is moved to target bin with the highest score for the move. During one iteration of the ILR, we go through all the modules in the placement region and follow the above steps for moving the modules. Subsequently, this iteration is repeated until there is no significant improvement in the wirelength.

In original FastPlace, during the first step of ILR the width and the height of each bin is set to 5*that of bin used during Cell Shifting. Such large bins are constructed to have a global view of the current placement and enable modules to move over long distance. During the subsequent steps, the width and height of the bins are gradually brought down to the value used in the cell shifting step. As a result, the movement of the modules gets progressively localized.

In order to do refinement on a bin level which is generated by MCF based approach, we set the bin size of ILR the same with that in MCF based approach. Then ILR can move cells for long distance.

4.2 Slice Based Refinement

Though ILR is very effective in reducing the wirelength, it is very slow. In order to have a faster technique to do the refinement, we proposed another new refinement technique called slice based refinement.

The main idea is to reorder the cells in a more global manner. In FastDP, they proposed a reordering technique called Local Reordering to reorder the cells within one standard cell row. Local Reordering is a technique to fix local horizontal errors. For any consecutive cells within a segment, local reordering tries all possible left-right ordering of cells and pick the order giving the best wirelength. After reordering, the technique will put the cells evenly in new order.

Figure 4.2 Reordering Technique

Though local reordering use the actual wirelength in the algorithm, it is slow and only do a refinement at a very local level. For example in Fig.4.2, Cell0, cell1 and cell2 may swap their order.But cell0 and cell 5 will not have the chance swap their order.

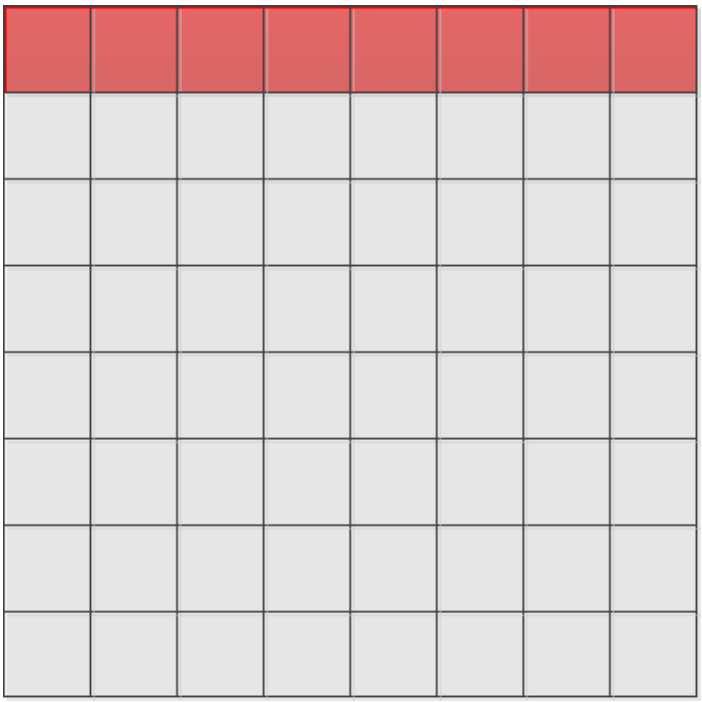

Figure 4.3 Row Refinement

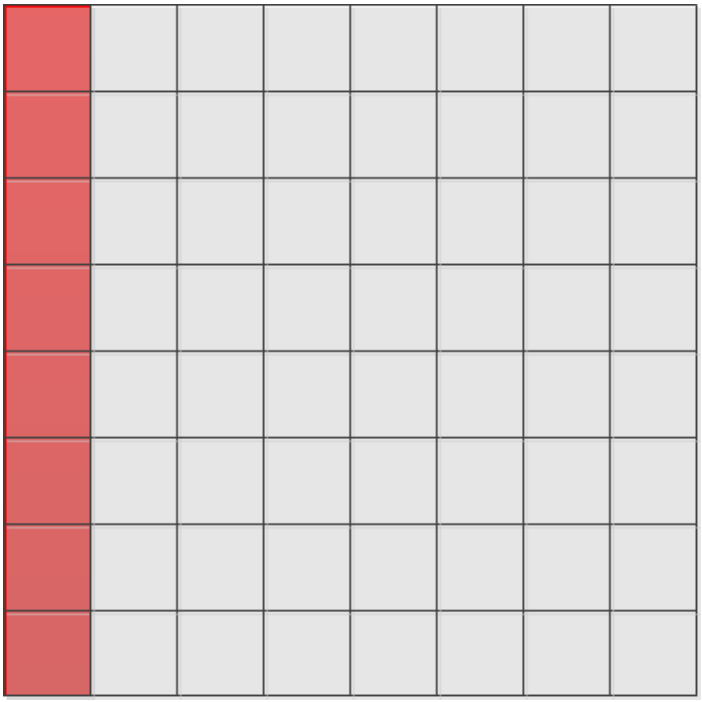

Figure 4.4 Column Refinement

In order to have a more fast and global way to reorder the cells, we proposed slice based refinement technique. In this technique, we divide the placement region into N

rows like Fig.4.3. We will then reorder cells row by row in a topological order until there is no significant improvement. And then we divide the placement region into N columns like Fig.4.4 and reorder cells column by column at the same way.

Since the algorithms for x direction and y direction are the same. Now we will only illustrate the algorithm in x direction. The detailed flow is as follows:

1 Calculate the current utilization of each bins

2 Repeat

3 Traverse each row i

4 Calculate optimal positions of cells in row i

5 Sort the cells in ascending order by optimal positions.

6 Pack cells one by one according to original utilization of each bin.

7 Until there is no significant wirelength improvement

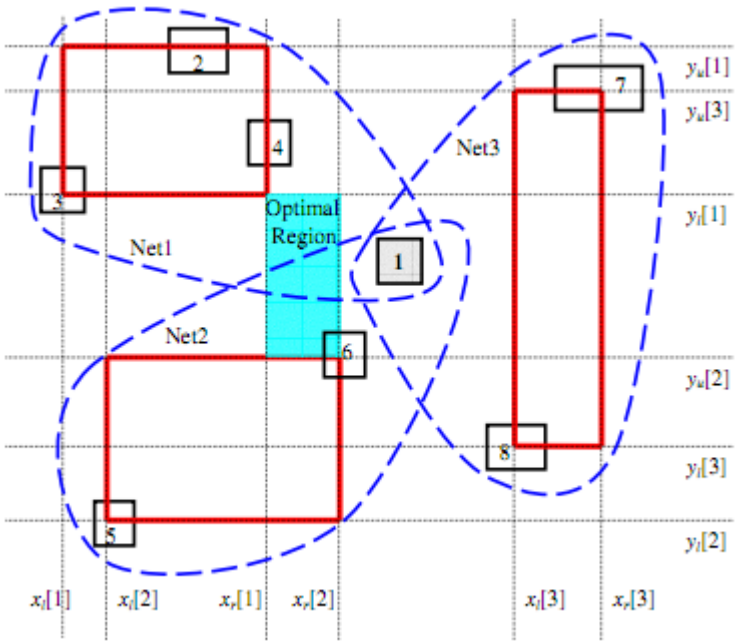

Figure 4.5 Optimal Region

In this algorithm, we will reorder cells row by row. For a particular cell in a row i,

we will fixed all other cells, and calculate the optimal region(Fig.4.5. And then sort the cells in row i in the ascending order by their center of their optimal region. After sorting, pack cells one by one from left to right according to the original utilization of each bin. For example in Fig.4.6, we have 4 cells in this row. Assume they have the same cell area. We divide the row in 4 bins. The first bin has the biggest utilization, the second and third bin has no cells. Then after reordering, we pack the cells according to the original utilization of bins as Fig.4.7 show.

Figure 4.6 Cell Distribution before Refinement

Figure 4.7 Cell Distribution after Refinement

The slice based refinement technique is much faster than ILR technique. But the disadvantage is that it is not as accurate as ILR. If the initial placement has good quality, this technique can't improve the wirelength any more.

In the first several iterations, the placement is very rough and have great room to improve for refinement, therefore we use Global Slice based refinement technique to do refinement instead of ILR which can reduce the runtime a little bit but not violating the quality.

Fig.4.8, Fig.4.9 and Fig.4.10 show the cell distribution of adaptec2 benchmark after initial quadratic program, MCF based approach for rough legalization and slice based technique. We can clearly see that after rough legalization, within some bins, the cells

are not distributed evenly. Some bins have a lot cell clusters. By refinement technique, the cells within a bin are more evenly distributed.

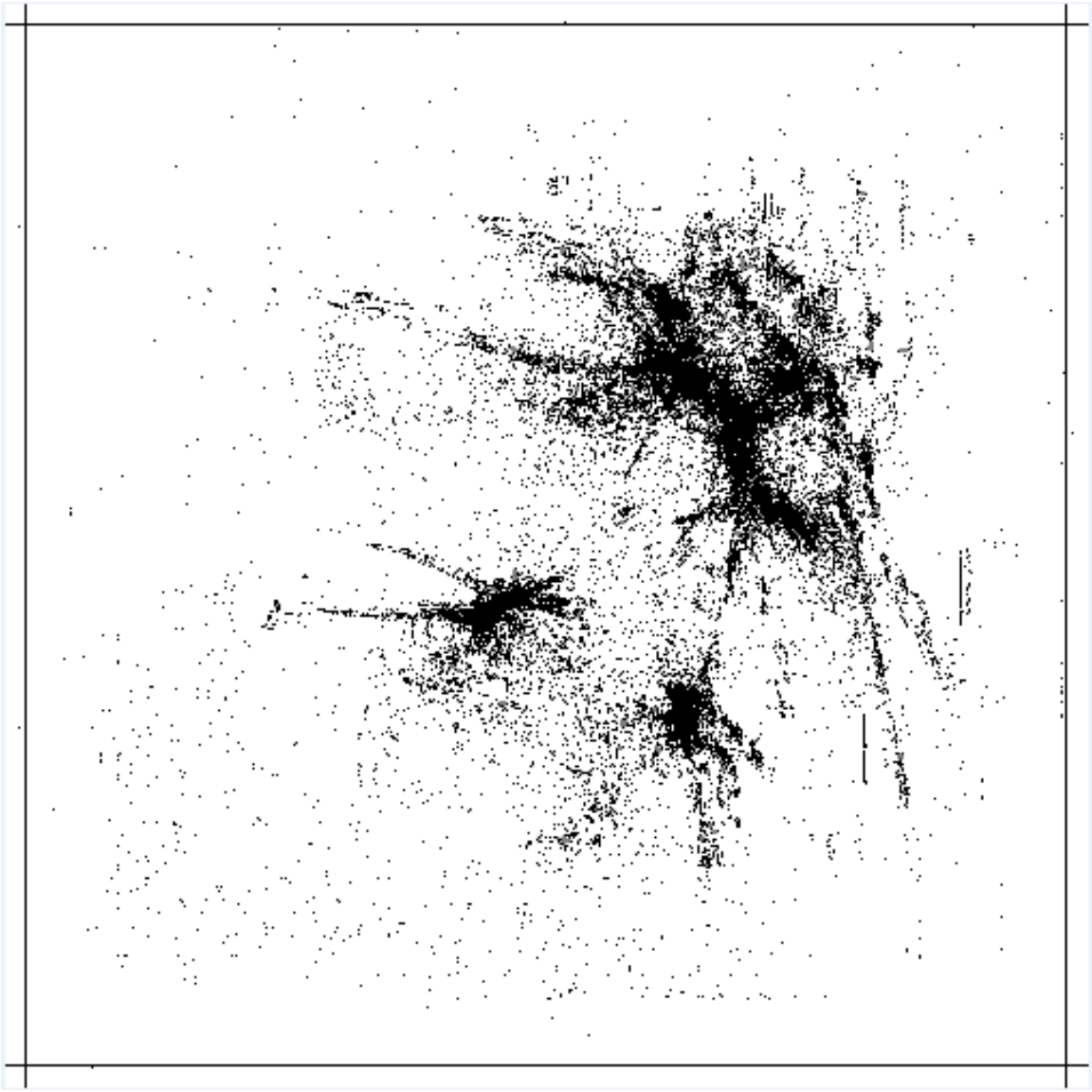

Figure 4.8 Cell Distribution after Quadratic Program

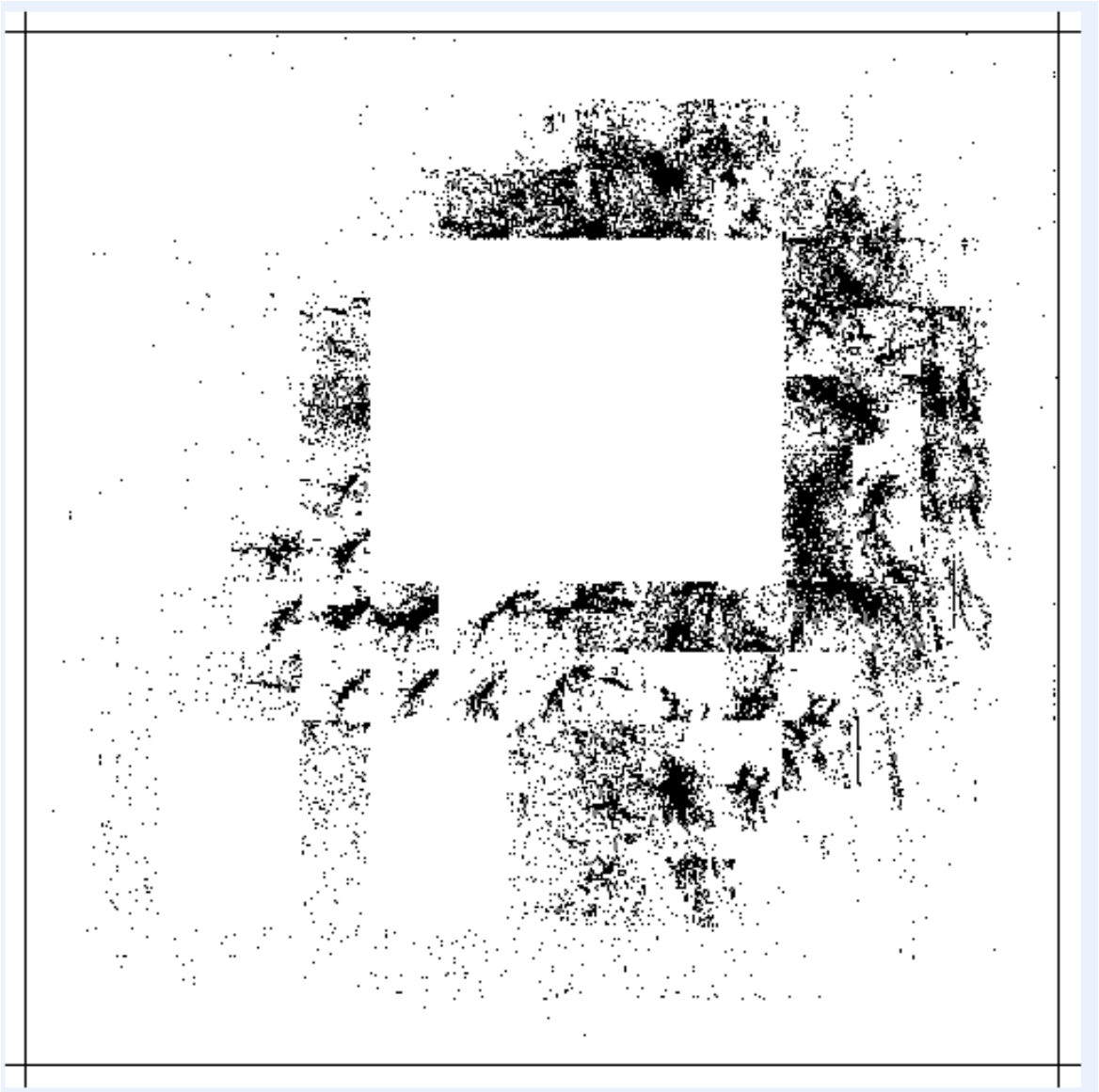

Figure 4.9 Cell Distribution after MCF based Rough Legalization

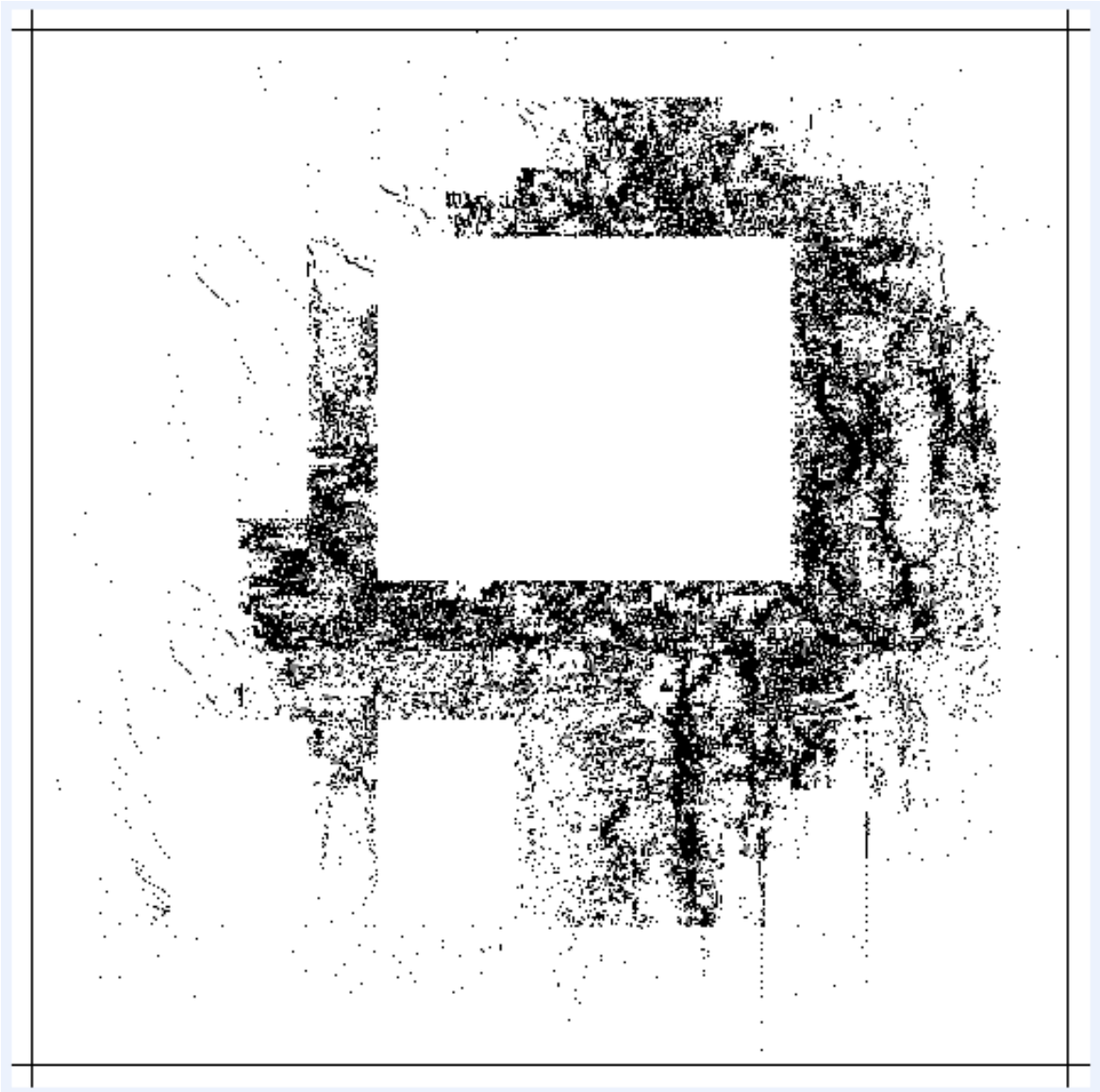

Figure 4.10 Cell Distribution after Slice based Refinement

CHAPTER 5. Experimental Results

Among many state-of-the-art placers, MAPLE, SimPL, RQL, Fastplace3, mPL6 are the best 5 placer in terms of quality. And SimPL and Fastplace3 are the fastest two placers.

Table 5.1, 5.2 and 5.3 are the comparison of our placer with others.

We can see from Table 5.1 that our placer is 1% better than RQL[16],1.5% than SimPL[3], 3.5% better than mPL6[22], 4% better than FastPlace3[23] in terms of wire-length for ISPD05 benchmark suites. MAPLE is placer with best quality among all the state-of-the-art placers. Our placer is 1.3% worse than MAPLE.

Table 5.1 HPWL(*10e6) comparison on the ISPD-2005 placement contest benchmark.

Case	Ours	Maple	RQL	SimPL	FastPlace3	mPL6
adaptec1	77.1	76.6	77.73	77.82	78.66	77.93
adaptec2	87	86.95	90.36	88.51	94.06	92.04
adaptec3	205	209.78	208	210.96	214.13	214.16
bigblue4	186	179.91	187.4	188.6	197.50	193.89
bigblue1	92.5	93.74	97.42	94.98	96.67	96.80
bigblue2	150	144.55	14.578	150.03	155.74	152.34
bigblue3	323.05	323.05	34.02	323.09	365.16	344.10
bigblue4	798	775.71	80.8	797.66	836.20	829.44
Comparison	1	0.99	1.015	1.01	1.040	1.035

Table 5.2 is the result of ISPD2006, we can see that our placement algorithm is still

better than other placement algorithm except MAPLE. The result of benchmark newblue1 is not promising, since our algorithm currently does not handle movable macros,which will be our future research, in a very accurate way. With new technique to handle movable macros, our placement algorithm will have better performance on ISPD2006,

Table 5.2 HPWL(*10e6) comparison on the ISPD-2006 placement contest benchmark.

Case	Ours	Maple	RQL	FastPlace3	mPL6
adaptec5	425.2	407.33	443.28	477.8	431.14
newblue1	73	69.25	64.43	76.53	67.02
newblue2	190.1	191.66	199.6	192.9	200.93
newblue3	279	268.07	269.33	305.6	287.05
newblue4	285	282.49	308.75	305.5	299.66
newblue5	545	515.04	537.49	612.7	540.67
newblue6	516.8	494.82	515.69	521.6	518.7
newblue7	1065	1032.6	1057.79	1088.75	1082.92
Comparison	1	0.967	1.0001	1.064	1.01

On runtime, we compare those placers with our's on benchmark ISPD-2005. Table 5.3 shows that our placer is 1.5 times faster than RQL, 4.4 times faster than mPL6 and 4 times faster than MAPLE. MAPLE compared the Fastplace with other placers. Therefore, we run the Fastplace 3 and our algorithm on 64 bit Intel(R) Xeon(R) CPU X5550 @ 2.67GHz, and then compare the result with other placers according the results provided from MAPLE[4] paper.

Table 5.3 Runtime comparison on the ISPD-2005 placement contest benchmark.

Case	Ours	Maple	RQL	SimPL	FastPlace3	mPL6
Comparison	1X	4X	1.5X	0.58X	0.63X	4.4X

Overall, we can see, our placer is better than any current state-of-the-art placers except MAPLE in terms of quality. Though we are 1.3% worse than MAPLE on quality, our placer is 4 times faster than MAPLE.

CHAPTER 6. Conclusion

This thesis shows a new force-directed quadratic placement algorithm that employs rough legalization and refinement technique during early global placement stage. It can achieve high quality placement at low CPU time.

The new proposed MCF based approach can roughly legalize the placement at flow bin level. This method not only spreads out cells, but also takes wirelength into consideration while spreading cells. And slice based refinement and ILR refinement technique can further improve the placement quality. These techniques are the key to generate very accurate target positions of cells for the quadratic program.

We are confident that there is room for further potential improvement both in terms of quality and runtime. For example, we can employ B2B net model which is more accurate than our current Hybrid net model to further improve wirelength. We can also improve runtime by incorporating state-of-art clustering technique in our hMCFPlace placer.

Bibliography

[1] A.R. Agnihorti, S. Ono, C. L. M. Y. A. K.-C.-K. K. and Madden, P. (2005). Mixed block placement via fractional cut recursive bisection. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, pages 748–761.

[2] Brenner, U. and Struzyna, M. (2005). Faster and better global placement by a new transportation algorithm. *In ACM/IEEE Design Automation Conference (DAC)*, pages 591–596.

[3] chul Kim, M., jin Lee, D., Markov, I. L., and Johannes, I. M. F. (2012a). Simpl: An effective placement algorithm.

[4] chul Kim, M., Viswanathan, N., Alpert, C. J., Markov, I. L., and Ramji, S. (2012b). Maple: Multilevel adaptive placement for mixed-size designs.

[5] Eisenmann, H. and Johannes, F. (1998). Generic global placement and oorplanning. *In ACM/IEEE Design Automation Conference (DAC)*, pages 269–274.

[6] G.-J. Nam, S. Reda, C. A. P. V. and Kahng, A. (2006). A fast hierarchical quadratic placement algorithm. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, pages 678–691.

[7] Hu, B. and Marek-Sadowska, M. (2002). Far: Fixed-points addition and relaxation based placement. *In ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 161–166.

[8] Hu, B. and Marek-Sadowska, M. (2005). Multilevel fixed-point-addition-based vlsi placement. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 24(8):1188–1203.

[9] J.A. Roy, D.A. Papa, S. A. H. C. A. N.-J. L. and Markov, I. (2005). Capo:robust and scalable open-source min-cut oorplacer. *In ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 224–226.

[10] J.M. Kleinhans, G. Sigl, F. J. and Antreich, K. (2002). Gordian: Vlsi placement by quadratic programming and slicing optimization. *EEE Transactions on Computer-Aided Design of Circuits and Systems(CAD)*, pages 161–166.

[11] Kahng, A. and Wang, Q. (2005). Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, pages 734–747.

[12] Kennings, A. and Vorwerk, K. (2006). Force-directed methods for generic placement. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, pages 2076–2087.

[13] Lobel, A. (http://typo.zib.de/opt-long_projects/Software/Mcf/). Mcf - a network simplex implementation.

[14] Min Pan, Viswanathan, N. C. C. (2005). An efficient and effective detailed placement algorithm. *In ACM/IEEE International Conference Conference (ICCAD)*, pages 48–55.

[15] R.-S. Tsay, E. K. and Hsu, C.-P. (1988). Proud: A sea-of-gates placement algorithm. *ieeedesigntest*, pages 44–56.

[16] Spindler, P., S. U. and Johannes, F. (2007). Rql: Global placement via relaxed quadratic spreading and linearization. *ACM/IEEE Design Automation Conference(DAC)*.

[17] Spindler, P., S. U. and Johannes, F. (2012). Kraftwerk2a fast force-directed quadratic placement approach using an accurate net model. *Computer-Aided Design of Integrated Circuits and Systems*.

[18] Sun, W.-J. and Sechen, C. (1993). Efficient and effective placement for very large circuits. *In IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 170–177.

[19] T.-C. Chen, Z.-W. Jiang, T.-C. H. H.-C. C. and Chang, Y.-W. (2006). A high-quality mixedsize analytical placer considering preplaced blocks and density constraints. *In IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 187–192.

[20] T. Chan, J. C. and Sze, K. (2005). Multilevel generalized force-directed method for circuit placement. *In ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 185–192.

[21] T. Taghavi, X. Y. and Choi, B.-K. (2005). Dragon2005: Large-scale mixed-size placement tool. *In ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 245–247.

[22] Tony Chan, Kenton Sze, J. S. and Xie, M. (2005). mpl6: Enhanced multilevel mixed-size placement with congestion control.

[23] Viswanathan, N., M. P. and Chu, C. (2007). Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control.

[24] Viswanathan, N. and Chu, C. C.-N. (2005). Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. *EEE Transactions on Computer-Aided Design of Circuits and Systems(CAD)*, pages 722–733.

[25] W. Naylor, R. D. and Sha, L. (2001). Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer. *U.S. Patent 6301693*.