VEAP : Global Optimization based Efficient Algorithm for VLSI Placement

Kong Tianming, Hong Xianlong, Qiao Changge
Department of Computer Science and Technology, Tsinghua University
Beijing, 100084, P.R.China
e-mail: {kongtm, hongxl, qiaocg}@tiger.cs.tsinghua.edu.cn

Abstract -- In this paper we present a very simple, efficient while effective placement algorithm for Row-based VLSIs. This algorithm is based on strict mathematical analysis, and provably can find the global optima. From our experiments, this algorithm is one of the fastest algorithms, especially for very large scale circuits. Another point desired to point out is that our algorithm can be run in both wirelength and timing-driven modes.

I. Introduction

The acceptance of row-based design styles is considerably influenced by the quality and the speed of the available design tools. In this paper we present a new placement algorithm named VEAP, which has been successfully applied to all cell-based layout styles and particularly to large circuits.

As the first step in the physical design process, the task of placement is to calculate the positions of the cells. Its difficulty increases as the cell count grows. Therefore, the classical approach to VLSI placement is based on the divide-and-conquer paradigm. Important representatives of this approach are based on min-cut graph partitioning(e.g., [1]-[4]). However, min-cut algorithms like those of Kernighan and Lin[5] and Fiduccia and Mattheyses[6] are iterative improvement heuristics that depend on an initial partition. Some authors([4][7]) proposed modifications and reported improved results.

Recently, alternative algorithms that model the placement problems as a linear or nonlinear continuous optimization problem have been studied([8]-[15]). In contrast to the min-cut approach, geometric information about cell and chip dimensions and pin locations can be used directly. Usually no starting solution is needed and all cells are treated simultaneously. Some of these methods apply partitioning to recursively create smaller subproblems. However, they restrict the simultaneous optimization to the initial step.

Getting stuck at local optima is a major drawback of partitioning-based methods. Efforts have been made to deal with this problem, especially to improve the widely used min-cut procedure([1][2]). Today there are some algorithms based on quadratic programming optimization([16][19]). This kind of algorithms has a very useful property: it is provably good based on mathematical analysis. But they suffered from long running time and one drawback: pads have to be assumed fixed, thus not appropriate when applied to circuits with floating pads. Ritual's developers have never reported results for very large scale circuits. In GORDIAN, after cells have been assigned into a region, they cannot move into another, thus placement quality is overconstrainted.

ASP-DAC '97
0-89791-851-7$5.00

In past years, many algorithms based on simulated annealing technique or genetic algorithm were published. This catalog of algorithms could maintain very excellent quality of placement, while they suffered from long running time heavily. The newest algorithm is from W.J.Sun and C.Sechen([17][18]), which can be run rather fast. However in this paper we found that without the help from random optimization techniques, equal or even better placement quality in less running time is still possible.

In this paper, we present a novel placement algorithm based on quadratic programming, but differently from previous algorithms. Our algorithm VEAP has the following characteristics: 1) Excellent quality: cells can move from the region it's assigned into another region; 2) High efficiency: it has very simple iterative form, and few memory requirements; 3) Inherent parallelism: further speedup is still possible.

In the rest of this paper, a detailed description of the algorithm and experiment results are given. In Section II, the algorithm is outlined. The detailed discussion is given in Section III. Algorithm analysis is given in Section IV. In Section VI, the algorithm for final placement is given. Finally we give experiment results.

II. Outline Of The Procedure

The placement algorithm VEAP is composed of alternating and interacting global optimization and partitioning steps. The input of VEAP consists of a net list, a cell library, and the chip's geometry description. In the placement procedure, pad cells may be fixed or to be assigned.

The main loop of VEAP is formed by an iteration of global optimization and partitioning steps. In each step of global optimization, a mathematical programming problem is formed and solved. In the following partitioning step cells are assigned into subregions according to their positions. Two linear constrains are generated for each subregion. As can be seen, the partitioning operation generates a slicing tree, in which each node represents a region. The loop is repeated until the number of partitioning reaches a pre-defined constant. Thus we can get a global placement for each cell.

III. Global Placement

In each global optimization step, a mathematical programming problem is derived and solved. The solution is a global placement of the cells.

A. Problem Formulation

The objective function is based on nets' quadratic wirelength model. For a net *n*, its length is computed according to the

following equation:

$$L_n = \sum_{i,j \in C_n} \left(\left(x_i + \xi_{i,n} - x_j - \xi_{j,n}\right)^2 + \left(y_i + \eta_{i,n} - y_j - \eta_{j,n}\right)^2 \right) \quad \ldots(1)$$

where C_n denotes the set of cells connected by net n, $(\xi_{i,n}, \eta_{i,n})$ are the coordinates of a pin connected to net n relative to the center of coordinates (x_i, y_i) of its cell. For simplicity, we omit these coordinates in the rest of this paper.

The objective is to minimize the weighted sum of the quadratic wirelength of all nets:

$$\phi = \sum_{n \in N} L_n w_n \quad \ldots(2)$$

where

$$w_n = \frac{1}{2 \times |C_n| \times (|C_n| - 1)}$$

At the *l*th optimization level, the placement plane will be divided into 4^l regions, each containing a subset of cells. Let $\Re^l$ denote the index set of regions. If *r* is a region, we use τ_r to denote the set of cells contained in *r*, and use (μ_r, ν_r) to denote the coordinates of the center of the region *r*. Since a cell can reside in only one region, we use the $\Re_i$ to denote the region cell *i* reside in. Thus for each region *r*, we get two constraints on the global placement:

$$\sum_{i \in \tau_r} x_i \Big/ |\tau_r| = \mu_r, \quad \sum_{i \in \tau_r} y_i \Big/ |\tau_r| = \nu_r \quad \ldots(3)$$

In the rest of this paper, we name this kind of constraints dispersion constraints.

Combining the objective function and the constraints, we get a *linearly constrained quadratic programming problem(LQP)*.

B. Solution Method

We use *Lagrange Relaxation Method* to solve the LQP. Each dispersion constraint will have an associated factor α_r or β_r. Thus according to Lagrange Relaxation Method, we turn the original problem into a series of unconstrained quadratic programming problems:

$$\max_{\alpha_r, \beta_r \geq 0} \min_{x,y} \phi(x, y, \alpha_r, \beta_r)$$

$$\phi = \sum_{n \in N} L_n w_n + \sum_{r \in \Re^l} \left(\sum_{i \in \tau_r} x_i \Big/ |\tau_r| - \mu_r \right) \alpha_r + \sum_{r \in \Re^l} \left(\sum_{i \in \tau_r} y_i \Big/ |\tau_r| - \nu_r \right) \beta_r \quad \ldots(4)$$

Let $\frac{\partial \phi}{\partial x_i} = 0$, $\frac{\partial \phi}{\partial y_i} = 0$, we have:

$$\frac{\partial \phi}{\partial x_i} = \sum_{n \in N_i} w_n \sum_{j \in C_n} 2\left(x_i - x_j\right) + \alpha_{\Re_i} \Big/ |\tau_{\Re_i}| = 0$$

$$\frac{\partial \phi}{\partial y_i} = \sum_{n \in N_i} w_n \sum_{j \in C_n} 2\left(y_i - y_j\right) + \beta_{\Re_i} \Big/ |\tau_{\Re_i}| = 0$$

Therefore,

$$x_i = \left(\sum_{n \in N_i} w_n \sum_{j \in C_n} x_j - 1/2 * \alpha_{\Re_i} \Big/ |\tau_{\Re_i}| \right) \Big/ \sum_{n \in N_i} w_n \quad \ldots(5)$$

$$y_i = \left(\sum_{n \in N_i} w_n \sum_{j \in C_n} y_j - 1/2 * \beta_{\Re_i} \Big/ |\tau_{\Re_i}| \right) \Big/ \sum_{n \in N_i} w_n \quad \ldots(6)$$

According to the above equations, we now give the global optimization algorithm for one level:

1.start from an initial assignment of iopads;
2.set up initial data $\alpha_r = 0, \beta_r = 0$ *;*
3.for count=1 to k
4. for i=1 to the-number-of movable cells
5. update x_i, y_i *according to (5)(6);*
6.update lagrangian factor
7. $\alpha_r += \sum_{i \in \tau_r} x_i \Big/ |\tau_r| - \mu_r$, $\beta_r += \sum_{i \in \tau_r} y_i \Big/ |\tau_r| - \nu_r$
8.linear assignment of iopads;
9.If desired accuracy reached, return; else goto 2;

IV. ALGORITHM ANALYSIS

A. Convergence

The term "Convergence" has two-fold meaning: the convergence of the overall global optimization(Lagrange Relaxation Method) and the convergence of the iteration method to solve the sub-lagrangian problem.

The convergence of the overall global optimization algorithm is obvious since the problem is a convex programming problem. The convergence of the iteration method to solve the lagrangian is guranteed since in each iteration, the lagrangian strictly decreases. And because the objective is a convex function, global optima is found.

If we rewrite the objective in matrices, thus we have:

$$\phi(x,y) = 1/2 \; x^T Q_x x - b_x^T x + 1/2 \; y^T Q_y y - b_y^T y$$

where the vectors *x* and *y* denote the coordinates of the movable cells to be placed. The matrices Q_x, Q_y correspond to the quadratic form of the nets' wirelength function. Both are positive definite if all movable cells are connected to some fixed cells either directly or indirectly. This holds since all cells will be connected to iopads and iopads are assumed fixed during the iterations. Thus we can get the global optima when

$$x = Q_x^{-1} b_x, y = Q_y^{-1} b_y$$

It can be seen that the above algorithm is in fact *Gauss-Seidel* iteration method. In general, *Successive Over Relaxation* method(SOR) is faster than Gauss-Seidel iteration method. Thus we could give another method to update x_i, y_i based on SOR, that is:

$$x_i^{(k+1)} = \left(\sum_{n \in N_i} w_n \left(\sum_{j \in C_n, j<i} x_j^{(k+1)} + \sum_{j \in C_n, j>i} x_j^{(k)} \right) - 1/2 * \alpha_{\Re_i} \Big/ |\tau_{\Re_i}| \right) \Big/ \sum_{n \in N_i} w_n * \omega + (1 - w) * x_i^{(k)} \quad \ldots(7)$$

$$y_i^{(k+1)} = \left(\sum_{n \in N_i} w_n \left(\sum_{j \in C_n, j<i} y_j^{(k+1)} + \sum_{j \in C_n, j>i} y_j^{(k)} \right) - 1/2 * \alpha_{\Re_i} \Big/ |\tau_{\Re_i}| \right) \Big/ \sum_{n \in N_i} w_n * \omega + (1 - w) * y_i^{(k)} \quad \ldots(8)$$

where *w* is the relaxation factor.

B. Dynamic Assignment of IOPADS

In many algorithms based on quadratic programming, there exists an assumption: the matrices Q_x, Q_y are the same and definite positive. In fact, this only hold when all iopads are fixed. For example, if all iopads are floating, Q_x, Q_y are not definite positive, thus those algorithms may face difficulty. When iopads are semi-floating(floating only on one side), Q_x, Q_y are not the same.

It's clear that our algorithm doesn't need such assumption. The assignment of floating iopads are dynamically performed via a linear assignment procedure. In general, direct interconnection between iopads is insignificant, so a linear assignment algorithm suffices.

C. Complexity

Each step of iteration to solve the lagrangian takes time O(*n*), where *n* is the number of cells. In general, the number of iterations tends to have a constant upper limit. To solve the original problem means to solve a series of lagrangians; the number depends on how tightly the accuracy is set. In practical computation, a limit $n^{0.5}$ suffices. Thus the overall time complexity is O($n^{1.5}$).

The basic interconnection of cells needs memory O(*n*). Two linear arrays are needed to store the linear constraint equations. Thus the overall space complexity is O(*n*).

D. Highly possible parallelism

Another point is that the algorithm is inherently highly parallel. From(7)(8), we know that a cell's coordinate computation requires that the cells(*j*) whose indices are less than the index of the cell(*i*) in question and which are directly connected to the cell *i* have been updated before. We can transform these constraints in graphic scheme, thus get a directed computation constrained graph *G(V,E).* A vertex *i* in *V* will correspond to a cell *i*; there exists a directed edge $(i, j) \in E$, if and only if *j* is less than *i* and *j* and *i* are directly connected.

Thus based on this graph, a simple parallel computation scheme is easily acquired. For example, we color the circuit connection graph in *p* colors, then we can partition the cell set in *p* subsets. Since cells in the same subset has no direct connection relation, they can be updated independently. *p* is less than or equal to the maximal degree of cells in the connection graph plus one. In practical circuits, the number of a cell's pins is limited, *p* is limited. Thus in large circuits, the speedup due to parallelism is rather promising.

V. Final Placement

The result of the alternating global optimization and partitioning steps is a global placement. Since in this placement there are many overlapping cells, thus a final placement has to follow. In this final placement stage, cells have to be moved such as no overlapping exists and some specification is obeyed.

In standard cell designs, cells are of approximately the same height but sometimes of fairly differing widths. The chip area is determined by the widths of the channels between cell rows and by the lengths of the rows including feedthroughs for nets crossing the rows. The goal is to obtain narrow channels with equally distributed low wiring density and rows with equal length.

We adopt a linear assignment algorithm, which tends to achieve minimal wirelength and even row length. We recursively partition the chip area until each region contains only tens of cells, in each of which cells will be assigned. This procedure is called "slot assignment".

In each region, the "slot assignment" problem is formulated as follows: given a set of slot positions and a cell set, to find a map between cells and slot positions such as the total wirelength is minimal. Assume that the number of slots are *m*, the number of cells to be assigned is *n* and the cost of assigning cell *i* to slot *j* is C_{ij} (estimated as the wire length when *i* is assigned to *j*). In general $n \le m$. Then the "slot assignment" problem is formulated as a linear assignment problem:

$$\text{minimize} \quad \sum_{ij} C_{ij} Z_{ij}$$

$$\text{subject to :}$$

$$\sum_{j=1}^{m} Z_{ij} = 1, \quad i = 1,\ldots,n$$

$$\sum_{i=1}^{n} Z_{ij} \le 1, \quad j = 1,\ldots,m$$

$$Z_{ij} \in \{0,1\}, \quad i = 1,\ldots,n, j = 1,\ldots,m$$

$x_i = \sum_{j=1}^{m} Z_{ij} X_j, \quad y_i = \sum_{j=1}^{m} Z_{ij} Y_j, \quad (i = 1,\ldots,n)$ is the position of cell *i*.

The general "slot assignment" which is a quadratic-assignment problem is difficult to solve. The presence of the above formulation is due to ignorance of the interconnection between cells in the same region. This may introduce some errors. In order to eliminate such errors and not to over-constraint cells, we may allow cells to migrate outside their assigned regions. This can be achieved by shifting *x*- and *y*-regions in such a way that adjacent regions are overlapping by half the region size. And the "slot assignment" algorithm will be run several times.

VI. Experiment Results

We've mentioned that our algorithm is very suitable for very large scale circuits. We can image that for small circuits, if the matrix *Q* is sparse, thus the direct inverse matrix(*DIMC*) computation will be faster than our algorithm. Fortunately, for practical circuits the matrix is rather sparse, thus *DIMC* is well suitable for some middle-scale circuits. For very large scale circuits, the matrix is still sparse, but due to the scale, *DIMC* cannot work well. This has been confirmed in our experiments.

We tested ten circuits. Their characteristics are listed in Table.I.

In Table.II, we compare our results with those obtained by Ritual. The comparisons are performed in terms of circuit area after slot assignment, the wiring length and CPU needed by the

placement methods measured on a Sun Sparc-20/50 workstation running SunOs4.1.4. Since Ritual is a timing-driven placement algorithm, in order to make a fair comparison, we run Ritual in wirelength mode.

From experimental results it's clear that our results are better than Ritual's. For circuit avq, our algorithm is the fastest ever reported.

REFERENCES

[1] U.Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," in *ACM/IEEE Proc. 16th DAC*, 1979, pp.1-10

[2] A.E.Dunlop and B.W.Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. CAD*, vol.CAD-4, pp.92-98, Jan.1985

[3] D.P.LaPotin and S.W.Director, "Mason: A global floorplanning approach for VLSI design," *IEEE Trans. CAD*, vol.CAD-5, pp.477-489, Oct.1986

[4] P.R.Suaris and G.Kedem, "An algorithm for quadrisection and its application to standard cell placement," *IEEE Trans. Circuits Syst.*, vol.35, pp.294-303, Mar.1988

[5] B.W.Kernighan and S.Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, pp.291-307,1970

[6] C.M.Fiduccia and R.M.Mattheyses, "A linear-time heuristic for improving network partitions," in *ACM/IEEE Proc. 19th DAC*, 1982, pp.175-181

[7] T.-K. Ng, J.Oldfield, and V.Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. CAD*, ICCAD-87, 1987, pp.470-473

[8] K.J.Antreich, F.M.Johannes, and F.H.Kirsch, "A new approach for solving the placement problem using force models," in *IEEE Int. Symp. Circuits and Systems*, Proc.ISCAS, 1982, pp.481-486

[9] G.J.Wipfler, M.Wiesel, and D.A.Mlynski, "A combined force and cut algorithm for hierarchical VLSI layout," in *ACM/IEEE Proc. 19th DAC*, 1982, pp.671-676

[10] C.-K.Cheng and E.S.Kuh, "Module placement based on resistive network optimization," *IEEE Trans. CAD*, vol.CAD-3, pp.218-225, July 1984.

[11] K.M.Just, J.M.Kleinhans, and F.M.Johannes, "On the relative placement and the transportation problem for standard-cell layout," in *ACM/IEEE Proc.23rd DAC*, 1986, pp.308-313

TABLE I

CIRCUIT CHARACTERISTICS

circuit	#pin	#cell	#net	#row
C2	373	590	963	9
sioo	62	602	664	12
balu	100	701	801	10
C5	301	1586	1887	16
C7	315	2150	2465	16
s13207	1490	4267	5757	24
s2	1608	9906	11514	28
c213	1489	11030	12519	20
s3	1726	15545	17271	24
avq	64	21854	22183	65

[12] R.Tsay, E.S.Kuh, and C.-P.Hsu, "PROUD: A fast sea-of-gates placement algorithm," in *ACM/IEEE Proc.25th DAC*, 1988, pp.318-323

[13] R.H.J.M.Otten, "Eigensolutions in top-down layout design," in *IEEE Int.Symp. on Circuits and Systems*, Proc. ISCAS, 1982, pp.1017-1020

[14] J.P.Blanks, "Near-optimal placement using a quadratic objective function," in *ACM/IEEE Proc. 22nd DAC*, 1985, pp.609-615

[15] J.Frankle and R.M.Karp, "Circuit placements and cost bounds by eigenvector decomposition," *IEEE Int. Conf. on CAD*, ICCAD-86, 414-417, 1986

[16] J.M.Kleinhans, G.Sigl, F.M.Johannes, and K.J.Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. CAD*, vol.CAD-10, no.3, 1991, pp.356-365

[17] W.Swartz and C.Sechen, "New algorithm for the placement and routing of macro cells," *Proc. Int. Conf. on CAD*, 1990, pp.336-339

[18] W.J.Sun and C.Sechen, "Efficient and Effective Placement for Very Large Circuits," *Proc. Int. Conf. on CAD*, 1993, pp.170-177

[19] A. Srinivasan, K. Chaudhary, E. S. Kuh, "RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs," *Proc. ICCAD*, November 1991, pp.48-51.

TABLE II

COMPARISON RESULTS

Circuit	Comparison	Ritual	Veap
C2	Wire	4.15×10^5	4.09×10^5
	Cpu Time	47.46	57.00
C5	Wire	1.09×10^6	1.03×10^6
	Cpu Time	194.04	170.39
C7	Wire	1.81×10^6	1.64×10^6
	Cpu Time	199.74	220.26
s13207	Wire	6.51×10^6	6.37×10^6
	Cpu Time	577.21	658.05
balu	Wire	2.74×10^7	2.67×10^7
	Cpu Time	296.89	320
s2	Wire	1.55×10^7	1.46×10^7
	Cpu Time	2617.76	2165
sioo	Wire	2.90×10^7	2.76×10^7
	Cpu Time	296.89	320
c213	Wire	1.60×10^7	1.25×10^7
	Cpu Time	3915.56	2863
s3	Wire	2.57×10^7	2.33×10^7
	Cpu Time	6740.73	4716
avq	Wire	NA	1.27×10^7
	Cpu Time	NA	7268.73