FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model

Natarajan Viswanathan, *Member, IEEE,* and Chris Chong-Nuen Chu, *Member, IEEE*

***Abstract*— In this paper, we present *FastPlace* – a fast, iterative, flat placement algorithm for large-scale standard cell designs. *FastPlace* is based on the quadratic placement approach. The quadratic approach formulates the wirelength minimization problem as a convex quadratic program that can be solved efficiently by some analytical techniques. However it suffers from some drawbacks. First, the resulting placement has a lot of overlap among cells. Second, the resulting total wirelength may be long as the quadratic wirelength objective is only an indirect measure of the linear wirelength. Third, existing net models tend to create a lot of non-zero entries in the connectivity matrix that slows down the quadratic program solver. To handle the above problems we propose: (1) An efficient *Cell Shifting* technique to remove cell overlap from the quadratic program solution and also accelerate the convergence of the solver. This technique produces a global placement with even cell distribution in a very short time. (2) An *Iterative Local Refinement* technique to reduce the wirelength according to the half-perimeter measure. (3) A *Hybrid Net Model* that is a combination of the traditional clique and star models. This net model greatly reduces the number of non-zero entries in the connectivity matrix and results in a significant speedup of the solver. Experimental results show that *FastPlace* is on average 13.4×, 102× and 19.9× faster than state-of-the art academic placers *Capo*, *Dragon* and *Gordian-Domino* respectively on a set of *IBM* benchmarks.**

***Index Terms*— Computer-aided design, analytical placement, standard cell placement, net models.**

I. Introduction

In recent years the role of placement in the physical design of large chips has grown dramatically [1], [2]. The main reason is that placement of circuit modules determines to a large extent interconnect length, and hence interconnect delay and routing resource demand. Interconnect delay has become the determining factor of circuit performance in present day Integrated Circuits. Hence, placement has become a major contributor to timing closure results. Current circuits often contain over a million placeable components, and it is predicted that circuit sizes will continue to double every three years [3]. Also, Cong et al. [4], [5] showed that existing placement algorithms are not scalable and stable. Therefore, it is likely that existing approaches may not be able to handle future circuits much larger in size. Hence, it is very essential to have extremely efficient placement algorithms.

Over the last few years, many placement algorithms have been proposed to handle the objective of wirelength minimization. These algorithms apply various approaches including *analytical placement* [6]–[13], *simulated annealing* [14], [15], and *partitioning/clustering* [16]–[18]. Analytical placement is a very promising approach for fast placement algorithm design. Analytical placement algorithms commonly utilize a quadratic wirelength objective function. Although the quadratic objective is only an indirect measure of the wirelength, its main advantage is that it can be minimized quite efficiently. As a result, analytical placement algorithms are relatively efficient in handling large problems. They typically employ a flat methodology so as to maintain a global view of the placement problem [7]–[11], [13]. For simulated annealing and partitioning/clustering based approaches, a hierarchical methodology is almost always employed to reduce the problem size and speed up the resulting algorithms [14]–[18]. Note that, when the placement problem is so large that a flat analytical approach cannot handle it effectively, a hierarchical analytical approach is beneficial. One way to convert to a hierarchical approach is to incorporate the fine granularity clustering technique proposed by Hu et al. [19]. This technique essentially introduces a two-level hierarchy to reduce the size of large-scale placement problems.

A major concern with the quadratic objective is that it results in a placement with a large amount of overlap among cells. Also, the quadratic objective by itself does not give the best possible wirelength. To handle these problems, Kleinhans et al. [10] used a placement-based bisection technique to recursively divide the circuit and add linear constraints to pull the cells in each partition to the center of the corresponding region. The FM [20] min-cut algorithm was used to improve the bisection and hence the wirelength. Vygen [13] applied a position-based quadrisection technique instead. A splitting-up technique to modify the netlist was also proposed to ensure that cells will stay in the assigned region. This technique also breaks down long nets and hence makes the objective behave like a linear function to some extent. Eisenmann et al. [7] introduced additional constant forces to each cell based on cell distribution to pull cells away from dense regions. Etawil et al. [8] added repelling forces for cells sharing a net to maintain a target distance between them and attractive forces by fixed dummy cells to pull cells from dense to sparse regions. Hu et al. [9] introduced the idea of fixed-point as a more general way to add forces for cell spreading. Hur et al. [12] used the spreading force of [7] to direct and control the ripple move optimization of Mongrel [21] to spread the cells. Kahng et al. [6] combined the cell spreading objective of [22] with a wirelength objective to achieve simultaneous cell spreading and wirelength optimization.

In this paper, we present a fast, iterative, flat placement algorithm called *FastPlace* for large-scale standard cell designs. *FastPlace* is based on the quadratic placement approach. The main contributions of our work are:

- An efficient *Cell Shifting* technique to remove cell overlap and accelerate the convergence of the quadratic program solver. The cell shifting technique roughly maintains the relative order of the cells in both horizontal and vertical directions as we believe that the quadratic objective function can determine a proper cell ordering. Hence, a high-quality global placement with even cell distribution can be produced in a short time.
- An *Iterative Local Refinement* technique to reduce the wirelength according to the half-perimeter measure. This technique is interleaved with Cell Shifting and Global Optimization during the final iterations of global placement. It makes use of the wirelength and cell distribution information provided by a coarse global placement and hence is very effective.
- A *Hybrid Net Model* that is a combination of the traditional clique and star [23] net models. We prove the equivalence of the Hybrid Net Model to the clique and star models. On average, the Hybrid Net Model results in a 2.95× reduction in the number of non-zero entries in the connectivity matrix as compared to

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (email: nataraj@iastate.edu; cnchu@iastate.edu).

the clique model. Consequently, it results in a 1.5× speed-up of the quadratic program solver.

The rest of the paper is organized as follows: Section II provides an overview of the algorithm. Section III describes the Global Optimization step. Section IV describes the Hybrid Net Model. Section V describes the Cell Shifting technique. Section VI describes the Iterative Local Refinement technique and Section VII describes the Detailed Placement technique. Experimental results are presented in Section VIII followed by Conclusions in Section IX.

II. Overview of the Algorithm

FastPlace essentially consists of three stages. The aim of the first stage is to simultaneously minimize the wirelength and spread the cells over the placement region to obtain a coarse global placement. It is composed of an iterative procedure in which we alternate between Global Optimization and Cell Shifting. Global Optimization involves minimizing the quadratic objective function. During Cell Shifting, the entire placement region is divided into equal sized bins and the utilization of each bin is determined. This gives a measure of the current placement distribution. The cells are then shifted around the placement region based on their respective bins and its current utilization. Finally, a spreading force is added to the cells to account for their movement during shifting. This is done to prevent the cells from collapsing back to their original positions during the next Global Optimization step.

The second stage is to refine the global placement by interleaving an Iterative Local Refinement technique with Global Optimization and Cell Shifting. This is done during the final stages of global placement. The Iterative Local Refinement technique is employed to reduce the wirelength based on the half-perimeter measure and to speed up the convergence of the algorithm. This stage of global placement yields a very well distributed placement solution with a very good value for the total wirelength.

The third stage is that of Detailed Placement. This consists of legalizing the current placement by assigning cells to pre-defined rows in the placement region. Within each row, the cells are then assigned to legal positions and any overlap among them is removed. It also consists of further reducing the wirelength by a greedy heuristic.

The algorithm *FASTPLACE* is summarized in Figure 1 and the individual components of the flow are discussed in more detail in Sections III–VII.

Algorithm *FASTPLACE*

Stage 1: Coarse Global Placement (CGP)

1. **Repeat**
2. Perform Global Optimization.
3. Perform Cell Shifting and Add Spreading Forces.
4. **Until** the placement is roughly even.

Stage 2: Wirelength Improved Global Placement (WIGP)

1. **Repeat**
2. Perform Global Optimization.
3. Perform Iterative Local Refinement.
4. Perform Cell Shifting and Add Spreading Forces.
5. **Until** the placement is very even.

Stage 3: Detailed Placement (DP)

1. **Repeat**
2. Further reduce wirelength using a greedy heuristic.
3. Legalize the current placement solution.
4. **Until** no significant improvement in wirelength.

Fig. 1. The *FASTPLACE* algorithm.

III. Global Optimization

This section describes the quadratic programming step of global placement refered as Global Optimization, which is the terminology used in [10]. The quadratic placement approach uses springs to model the connectivity of the circuit. The total potential energy of the springs, that is a quadratic function of their length, is minimized[1] to produce a placement solution. In order to model the circuit by a spring system, each multi-pin net needs to be transformed into a set of two-pin nets by a suitable net model. In the following, we assume that this transformation has been applied. The net model used will be discussed in Section IV.

Let n be the number of movable cells in the circuit and (x_i, y_i) the coordinates of the center of cell i. A placement of the circuit is given by the two n-dimensional vectors $\mathbf{x} = (x_1, x_2, .., x_n)$ and $\mathbf{y} = (y_1, y_2, .., y_n)$. Consider the net between two movable cells i and j in the circuit. Let W_{ij} be its weight. Then the cost of the net between the cells is:

$$\frac{1}{2} W_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \quad (1)$$

[1]Equivalently, a force equilibrium state of the spring system is found.

If a cell i is connected to a fixed cell f with coordinates (x_f, y_f), the cost of the net is given by:

$$\frac{1}{2} W_{if}[(x_i - x_f)^2 + (y_i - y_f)^2] \quad (2)$$

The objective function that sums up the cost of all the nets can be written in matrix notation as [24]:

$$\Phi(x, y) = \frac{1}{2} x^T Q x + d_x^T x + \frac{1}{2} y^T Q y + d_y^T y + \text{constant} \quad (3)$$

where Q is an $n \times n$ symmetric positive definite matrix and d_x, d_y are n-dimensional vectors. Since equation (3) is separable into $\Phi(x, y) = \Phi(x) + \Phi(y)$, only the the x-dimension is considered for subsequent discussion, which is:

$$\Phi(x) = \frac{1}{2} x^T Q x + d_x^T x + \text{constant} \quad (4)$$

Let q_{ij} be the entry in row i and column j of matrix Q. From expression (1), the cost in the x-direction between two movable cells i and j is $\frac{1}{2} W_{ij}(x_i^2 + x_j^2 - 2x_i x_j)$. The first and second terms contribute W_{ij} to q_{ii} and q_{jj} respectively. The third term contributes $-W_{ij}$ to q_{ij} and q_{ji}. From expression (2), the cost in the x-direction between a movable cell i and a fixed cell f is $\frac{1}{2} W_{if}(x_i^2 + x_f^2 - 2x_i x_f)$. The first term contributes W_{if} to q_{ii}. The third term contributes $-W_{if} x_f$ to the vector d_x at row i and the second term contributes to the constant part of equation (4). The objective function (4) is minimized by solving the system of linear equations represented by:

$$Qx + d_x = 0. \quad (5)$$

Equation (5) gives the solution to the unconstrained problem of minimizing the quadratic function in (4). In *FastPlace*, we solve such an unconstrained minimization problem throughout the placement process. We do not add any constraint to the problem formulation. This is because the spreading forces added during Cell Shifting are produced by pseudo nets connecting the cells to the chip boundary. This only introduces some terms in the form of expression (2) and causes some changes to the diagonal of matrix Q and the vector d_x as described above.

IV. Hybrid Net Model

To handle the large placement problem size, a fast and accurate technique is needed to solve equation (5). Since matrix Q is sparse, symmetric and positive definite, we solve equation (5) by the preconditioned Conjugate Gradient method. The Incomplete Cholesky Factorization of matrix Q is used as the preconditioner [25], [26]. The runtime of the solver is directly proportional to the number of non-zero entries in matrix Q. This in turn is equal to the number of two-pin nets in the circuit. Hence, it becomes imperative to choose a good net model so as to have minimal non-zero entries in matrix Q.

We propose a Hybrid Net Model that is a combination of the clique and star net models. We show experimentally in Section VIII that the Hybrid Net Model reduces the number of non-zero entries in matrix Q by 2.95× over the traditional clique model. In the subsequent discussion, we give a brief overview of the clique and star net models, and introduce the Hybrid Net Model. Then, we prove the equivalence of the clique and star models, and hence the consistency of the Hybrid Net Model.

A. Clique, Star and Hybrid Net Models

The clique model is the traditional model used in analytical placement algorithms. In the clique model, a k-pin net is replaced by $k(k-1)/2$ two-pin nets forming a clique. Let W be the weight of the k-pin net. Some commonly used values for the weight of the two-pin nets are $W/(k-1)$ (e.g., [13]) and $2W/k$ (e.g., [7], [10]). The clique model for a 5-pin net is illustrated in Figure 2(a).

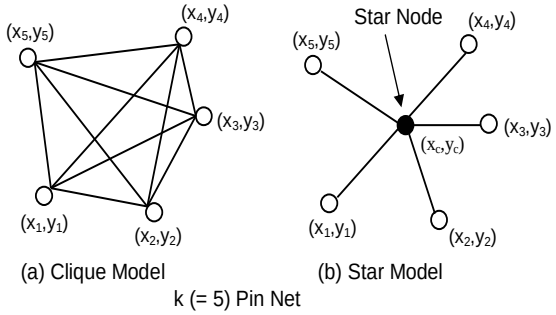

Fig. 2. Net models.

Recently, Mo et al. [23] utilized the star net model in a macro-cell placer. In the star model, each net has a star node to which all pins of the net are connected. Hence, a k-pin net will yield k two-pin nets. The star model for a 5-pin net is illustrated in Figure 2(b). Mo et al. [23] create a star node even for two-pin nets and point out that the clique model generates on average 30% more two-pin nets than the star model for the MCNC92 macro block benchmarks. Vygen [13] also switches to a star model for very large nets to reduce the number of terms in the objective function, but has not shown the validity of mixing the clique and star models in quadratic placement. In addition, neither paper has discussed the method to set the weight of the nets introduced by the star model.

In the following subsection we prove that for a k-pin net of weight W, if we set the weight of the two-pin nets introduced, to γW in the clique model and $k\gamma W$ in the star model for any γ, the clique model is equivalent to the star model in quadratic placement. Therefore, the two models can be used interchangeably.

We propose a Hybrid Net Model that uses a clique model for two-pin and three-pin nets, and a star model for nets with four or more pins. We set γ to $1/(k-1)$ in *FastPlace* as it works well experimentally. By using the star model for nets with four or more pins, we will generate much fewer two-pin nets and consequently fewer non-zero entries in the matrix Q than the clique model. By using the clique model for two-pin nets, we will not introduce one extra net and two extra variables (corresponding to the x and y dimensions) per two-pin net as in [23]. We choose to use the clique model for three-pin nets because it is better than the star model for the following reasons: First, if two cells are connected by more than one two-pin or three-pin net in the original netlist, the two-pin nets generated by the clique model between the two cells can be combined and will only introduce a single non-zero entry in the matrix Q. Second, it will not introduce an extra pair of variables.

B. Equivalence of the Hybrid Net Model to the Clique and Star Net Models

In this subsection, we show that the clique model is equivalent to the star model in quadratic placement if net weights are set appropriately. It follows that the clique, star and Hybrid net models are all equivalent.

Lemma 1: For any net in the star model, the star node under force equilibrium is at the center of gravity of all pins of the net.

Proof: Consider a k-pin net. Let x_s be the x-coordinate of the star node and let W_s be the weight of the two-pin nets introduced. Then the total force on the star node by all the pins is given by:

$$F = \sum_{j=1}^{k} W_s(x_j - x_s).$$

Under force equilibrium, the total force $F = 0$. Therefore,

$$x_s = \frac{\sum_{j=1}^{k} x_j}{k}. \tag{6}$$

Hence the lemma follows. ■

Theorem 1: For a k-pin net, if the weight of the two-pin nets introduced is set to W_c in the clique model and kW_c in the star model, the clique model is equivalent to the star model in quadratic placement.

Proof: For the clique model, the total force on a pin i by all the other pins is given by:

$$F_i^{clique} = W_c \sum_{j=1, j\neq i}^{k} (x_j - x_i) \tag{7}$$

For the star model, all the pins of the net are connected to the star node. The force on a pin i due to the star node is given by:

$$\begin{aligned} F_i^{star} &= kW_c(x_s - x_i) \\ &= kW_c\left(\frac{\sum_{j=1}^{k} x_j}{k} - x_i\right) \qquad \text{by Lemma 1} \\ &= W_c\left(\sum_{j=1}^{k} x_j - kx_i\right) \\ &= W_c \sum_{j=1, j\neq i}^{k} (x_j - x_i) \\ &= F_i^{clique} \end{aligned}$$

As the forces are same in both models for all pins, the lemma follows. ■

A combination of the clique and star models has been used in the industry and academia. Previously, the star model has been only used for high degree nets, so as to reduce the number of non-zero entries in matrix Q and speed-up the solver. But, the validity of mixing the clique and star models in quadratic placement has not

been proven. Also, there has been no mention about the method to set the weights of the two-pin nets introduced by the two models if they are combined. Gordian [10] also uses a star node to formulate the problem for multi-pin nets. But, to reduce the number of variables, they explicitly state that they substitute the co-ordinates of the star nodes with the mean values of the co-ordinates of the pins. In doing so, even though they have fewer variables, they still have the same number of non-zero entries in matrix Q as the traditional clique model.

In the star model used in our algorithm, we introduce two extra variables (one for each x and y dimension) in the matrix corresponding to the star node. By introducing these variables, even though the total number of variables has increased, the total number of non-zero entries in the matrix has been greatly reduced. Considering the case of a k-pin net, our approach will only introduce k non-zero entries in the matrix for the star model. Whereas, the approach followed in [10] will still introduce $k(k-1)/2$ non-zero entries in the matrix.

Also, in this paper, we have described the method to set the weights of the two-pin nets introduced by the clique and star models. Consequently, based on the weights of the two-pin nets, we have proven the equivalence of the two models and hence the validity of mixing them in quadratic placement. Based on the proof, the main novelty of our Hybrid Net Model is that we can use the star model even for nets with just four or more pins. We no longer have to restrict its usage to only high-degree nets. If a combination of the clique and star models are used, the Hybrid Net Model will give the minimum possible non-zero entries in matrix Q.

To the best of our knowledge, the aforementioned proof and treatment of the star model has not been reported in prior literature.

V. Cell Shifting

Global Optimization essentially minimizes the quadratic objective function. However, it does not consider the overlap among cells. Therefore, the resulting placement has a lot of cell overlap and is not distributed over the placement region. Cell Shifting evens out the placement by distributing the cells over the placement region while retaining their relative ordering obtained from the Global Optimization step. In the next sub-sections we describe the steps involved in Cell Shifting.

A. Calculation of Bin Utilization

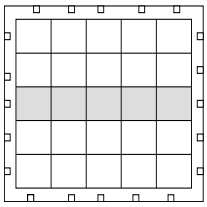

Fig. 3. Regular bin structure.

Initially, the placement region is divided into equal sized bins (Figure 3). Each bin can accommodate an average of 4 cells. Based on the placement obtained from Global Optimization, the utilization of each bin (U_i) is then computed. U_i is defined as the total area of all the cells inside bin i divided by the bin area. In calculating the total area of all the cells, we sum the areas of all cells completely covered by bin i and the overlap area between the bin and the cell for cells that partially overlap with bin i. The cells are then shifted around the placement region based on their respective bins and its current utilization.

B. Shifting of Cells

Let us consider the case where the cells are shifted in the x-dimension. To shift cells, we go through every row of the regular bin structure and move cells present in the row. Shifting of cells is a two step process. First, based on the current utilization of all the bins in a particular row an unequal bin structure reflecting the current bin utilization is constructed. Second, every cell belonging to a particular bin in the regular bin structure is then linearly mapped to the corresponding bin in the unequal bin structure. As a result of this mapping, cells in bins with a high utilization will shift in a way so as to reduce their utilization and the overlap among themselves. Once all the rows of the regular bin structure have been considered, we go through every column and shift the cells in the y-dimension by following the two steps mentioned above.

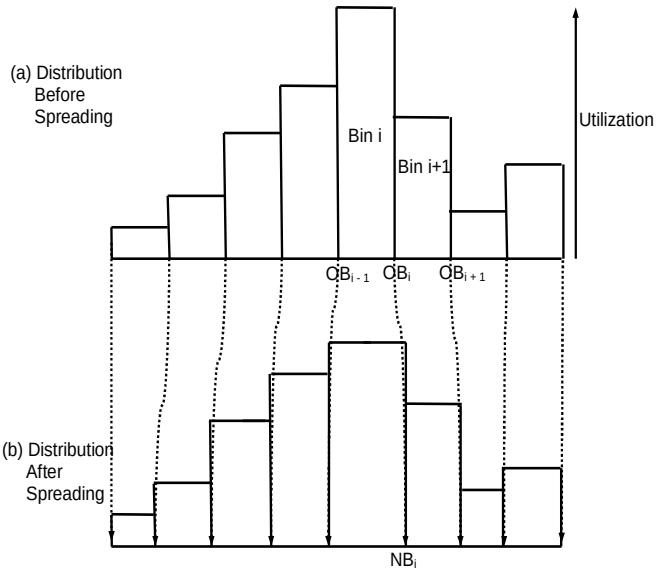

Fig. 4. (a) Regular bin structure (b) Unequal bin structure and utilization after shifting.

To illustrate the shifting in the x-dimension, consider a particular row in the regular bin structure (shaded row in Figure 3). The utilization of all the bins in this row is given in Figure 4(a). The unequal bin structure constructed from the regular bin structure is illustrated in Figure 4(b). To get the equation for the new bin structure, from Figure 4 let,

- OB_i: x-coordinate of the boundary of bin i corresponding to the regular bin structure
- NB_i: x-coordinate of the boundary of bin i corresponding to the unequal bin structure

Then,

$$NB_i = \frac{OB_{i-1}(U_{i+1}+\delta) + OB_{i+1}(U_i+\delta)}{U_i + U_{i+1} + 2\delta} \tag{8}$$

The idea behind Cell Shifting is to even out the utilization among adjacent bins. Hence, the intuition behind the above formula is to construct the new bin such that it averages the utilization of bin i and bin $i+1$. The reason for having the parameter δ is as follows: Let, $\delta = 0$ and $U_{i+1} = 0$, then from equation (8) it can be seen that, $NB_i = OB_{i+1}$ and $NB_{i+1} = OB_i$. This results in a cross-over of bin boundaries in the unequal bin structure that results in improper mapping of the cells. To avoid this problem, we need the parameter δ, that is set to a value of 1.5.

For performing the linear mapping of cells, If,

- x_j: x-coordinate of cell j in bin i before mapping (obtained from the Global Optimization step)
- $x\prime_j$: x-coordinate of cell j in bin i after mapping

Then,

$$\frac{x_j - OB_{i-1}}{OB_i - OB_{i-1}} = \frac{x\prime_j - NB_{i-1}}{NB_i - NB_{i-1}}$$

or,

$$x\prime_j = \frac{NB_i(x_j - OB_{i-1}) + NB_{i-1}(OB_i - x_j)}{OB_i - OB_{i-1}} \quad (9)$$

During the initial placement iterations a few bins in the placement region will have an extremely high bin utilization. Consequently, cells in these bins will have a tendency to shift over large distances. This will perturb the current placement solution by a large amount. This effect will get added over iterations and result in a final placement with a high value of the total wirelength. Therefore, to control the actual distance moved by any cell during shifting, we introduce two *movement control parameters*, α_x and α_y (< 1) for the x and y dimensions. α_x and α_y are increasing functions that are inversely proportional to the maximum bin utilization and have a very small value during the initial placement iterations. In the x-dimension say, once the position of cell j has been determined after mapping, the actual distance moved by the cell is $\alpha_x \mid x\prime_j - x_j \mid$.

Thus, the cells are shifted over very small distances during the initial placement iterations. During the final stages of global placement, the cells will be distributed quite evenly, and not have a tendency to shift over large distances. Then, α can take a larger value to accelerate convergence. The expressions for α_x and α_y are:

$$\alpha_y = 0.02 + \frac{0.5}{max(U_i)}$$

$$\alpha_x = 0.02 + \left(\frac{0.5}{max(U_i)}\right)\left(\frac{averageCellWidth}{cellHeight}\right)$$

We use the maximum utilization among all bins as a measure of the evenness of cell distribution. The lesser the maximum utilization, the more distributed are the cells. The maximum bin utilization can also be used as a measure of the efficiency of the Cell Shifting technique. Figure 5 shows the change in the maximum bin utilization value over placement iterations for the circuit ibm01. It can be seen that within 19 iterations of global placement the cells are spread out quite evenly over the placement region and the maximum bin utilization reaches the required threshold for us to begin detailed placement. This also shows that Cell Shifting is very effective in accelerating the convergence of the quadratic program solver. Correspondingly, Figure 6 gives the change in the wirelength over global placement iterations. This figure also includes the final wirelength obtained after detailed placement shown as iteration number 20. We can see a jump in the wirelength value between iterations 9 and 11. This is when the algorithm transitions from the CGP to the WIGP stage.

C. Addition of Spreading Forces

After the cells have been shifted in the x and y dimensions, additional forces need to be added to them so that they do not collapse back to their previous positions during the next Global Optimization step. This is achieved by connecting each cell to a corresponding pseudo pin added at the boundary of the placement region. The pseudo pin and pseudo net addition is illustrated in Figure 7.

Let (x_j^f, y_j^f) be the target position of cell j after Cell Shifting. When it is moved to the target position, it will experience a force due to its connectivity with the other cells or star nodes in the placement region. This force can also be viewed as the force required to move the cell from its original position (before Cell Shifting) to the target

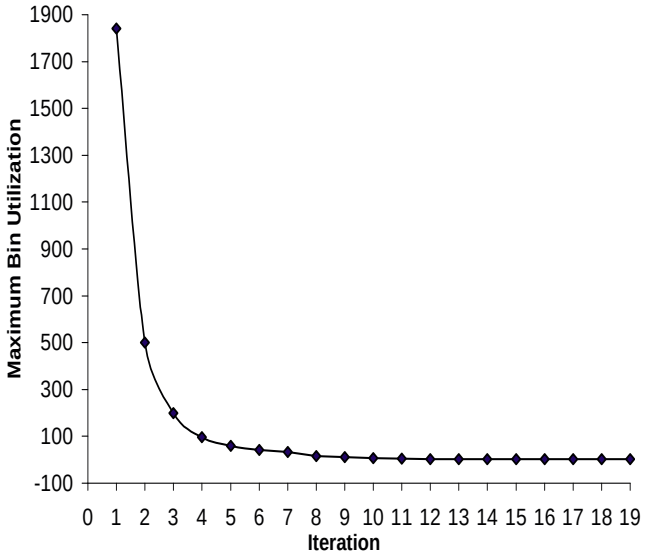

Fig. 5. Maximum bin utilization vs iteration number for circuit ibm01.

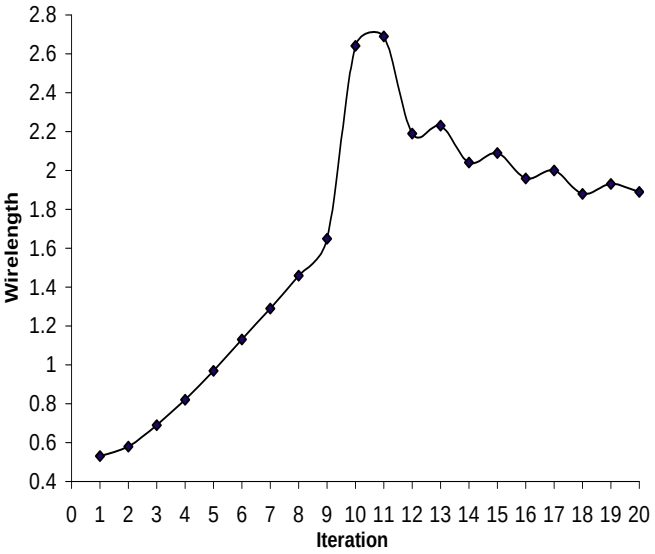

Fig. 6. Wirelength vs iteration number for circuit ibm01.

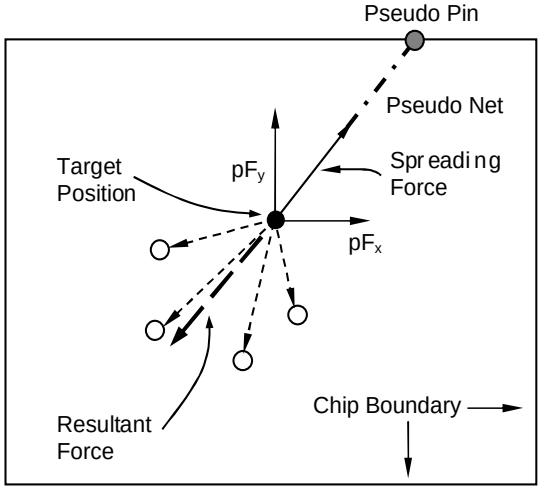

Fig. 7. Pseudo pin and pseudo net addition.

position. The spreading force added to the cell corresponds to this force experienced by the cell in its target position.

To illustrate the addition of the spreading force, consider Figure 7. When cell j (solid circle) is moved to its target position, it will experience a force due to the other cells connected to it (empty circles). When determining this force, we assume that all cells connected to cell j are still in their original positions (before Cell Shifting). The resultant force due to the cells connected to cell j is given by the "Resultant Force" vector. The spreading force has the same magnitude as the "Resultant Force" vector but is in the opposite direction.

To determine the position of the pseudo pin and the spring constant of the pseudo net, If,

- pF_x: x-component of the spreading force.
- pF_y: y-component of the spreading force
- pD_x: x-component of the distance between the pseudo pin and target position of cell j
- pD_y: y-component of the distance

Then, the position of the pseudo pin can be determined by the intersection of the "Spreading Force" vector with the chip boundary. A pseudo net for cell j is one that connects the cell from its target position to its pseudo pin. The spring constant for the pseudo net is given by $\beta = \sqrt{pF_x^2 + pF_y^2}/\sqrt{pD_x^2 + pD_y^2}$. During each iteration of Global Placement, a new spreading force and corresponding pseudo pin position is determined for every cell.

Since the pseudo pin is a fixed pin present at the boundary, we know from expression (2) and the subsequent analysis in Section III, that only the diagonal of matrix Q and the d_x and d_y vectors need to be updated for every cell. Hence, it takes only a single pass of $O(n)$ time, where n is the total number of movable cells in the circuit, to regenerate the connectivity matrix for the next Global Optimization step.

Thus we have incorporated an extremely fast Cell Shifting technique to distribute the cells over the placement region.

VI. Iterative Local Refinement

Since the quadratic objective function is only an indirect measure of the linear wirelength, it does not yield the best possible result in terms of wirelength. To offset this disadvantage, we incorporate an Iterative Local Refinement technique to further reduce the wirelength.

The Iterative Local Refinement technique is interleaved with the Global Optimization and Cell Shifting steps during the WIGP stage. This technique acts on a coarse global placement obtained from the previous stage and hence is very effective in minimizing the wirelength. Unlike other approaches, this technique uses the actual position of a cell and the half-perimeter bounding rectangle measure of all nets connected to the cell to move it around the placement region. The technique is based on a greedy heuristic that mainly tries to minimize the wirelength while trying to reduce the maximum bin utilization so as to speed-up the convergence of the algorithm.

A. Bin Structure

This technique also employs a regular bin structure to estimate the current utilization of a placement region for performing wirelength improvement. Cells are then moved from *source* to *target* bins based upon the wirelength improvement and target bin utilization. During the first iteration of the WIGP stage, the width and height of each bin for the Refinement is set to 5 times that of the bin used during Cell Shifting. Such large bins are constructed to enable cell movement over large distances. This is to minimize the wirelength of long nets that might span a large part of the placement area. The width and height of the bins are gradually brought down to the values used in the Cell Shifting step over subsequent iterations of the WIGP stage.

B. Description of the Technique

Once the utilization of all the bins in the placement region has been determined, we traverse through all the cells in the placement region and determine their respective *source* bins. For every cell present in a bin we compute four *scores* corresponding to the four possible cell movement directions. For calculating the score, we assume that a cell is moving from its current position in a *source* bin to the same position in a *target* bin that is adjacent to it. That is, we move the cell by one bin width. Each score is a weighted sum of two components. The first being the wirelength reduction for the move and the second being a function of the utilization of the source and target bins. For the first component, the wirelength is computed as the total half-perimeter of the bounding rectangle of all nets connected to the cell. Hence, it is much more accurate than the quadratic objective function. Since the Local Refinement technique is mainly used to reduce the wirelength, a higher weight is used for the first component. If all the four scores are negative, the cell will remain in the current bin. Otherwise, it will move to the target bin with the highest score for the move. During one iteration of the Local Refinement, we traverse through all the bins in the placement region and follow the above steps for cell movement. Subsequently, this iteration is repeated until there is no significant improvement in the wirelength. The Iterative Local Refinement technique is then followed by Cell Shifting wherein we add the spreading forces as described previously to reflect the current placement.

To judge the contribution of the Iterative Local Refinement technique on the overall runtime and wirelength, we ran two different flows of the algorithm: (a) the original flow incorporating the technique (b) without the technique. Table IV summarizes the total runtime and final wirelength results for the two flows. It can be seen that the flow without Iterative Local Refinement showed an average reduction of 32.3% in the total runtime, but resulted in a 15.1% increase in the final wirelength. Also, the increase in wirelength is more prominent with an increase in the circuit size. This shows that the Iterative Local Refinement technique is quite effective in reducing the wirelength of the placement.

VII. Detailed Placement

The Detailed Placement stage legalizes the solution obtained from global placement. It assigns all the standard cells to pre-defined rows in the placement region. Once the cells have been assigned to the rows any remaining overlap among them is removed and they are assigned to legal positions within the rows. During legalization, the detailed placement also tries to further reduce the wirelength by employing a technique similar to Iterative Local Refinement. The difference is that during detailed placement, the technique acts on cells that have been assigned to the actual rows of the placement region. Besides, it puts a higher weight on the utilization factor than the wirelength factor as the emphasis is on removal of overlap among cells to obtain a legalized placement.

VIII. Experimental Results

A. Benchmarks and other placers

FastPlace is implemented in C and has been tested on a set of benchmarks derived from the ISPD-02 IBM-MS Mixed-size Placement benchmark suite [27], [28] and the PEKO suite [4], [29]. The ISPD-02 IBM-MS benchmarks consist of macro blocks and hence had to be modified to be tested on *FastPlace*. The height of all the macro blocks was brought down to the standard cell height. The average width of all the modules in the original benchmark was computed and the width of all macros exceeding 4 times the average width was assigned to a value of $4\times$ average width. All designs in the

TABLE I
PLACEMENT BENCHMARK STATISTICS.

Ckt	#Nodes	#Pads	#Nets	#Pins	#Rows	Ckt	#Nodes	#Pads	#Nets	#Pins	#Rows
ibm01	12506	246	14111	50566	96	Peko01	12506	488	14111	50566	113
ibm02	19342	259	19584	81199	109	Peko02	19342	608	19584	81199	140
ibm03	22853	283	27401	93573	121	Peko03	22853	660	27401	93573	152
ibm04	27220	287	31970	105859	136	Peko04	27220	718	31970	105859	166
ibm05	28146	1201	28446	126308	139	Peko05	28146	732	28446	126308	169
ibm06	32332	166	34826	128182	126	Peko06	32332	784	34826	128182	181
ibm07	45639	287	48117	175639	166	Peko07	45639	932	48117	175639	215
ibm08	51023	286	50513	204890	170	Peko08	51023	984	50513	204890	227
ibm09	53110	285	60902	222088	183	Peko09	53110	1004	60902	222088	231
ibm10	68685	744	75196	297567	234	Peko10	68685	1144	75196	297567	263
ibm11	70152	406	81454	280786	208	Peko11	70152	1154	81454	280786	266
ibm12	70439	637	77240	317760	242	Peko12	70439	1156	77240	317760	266
ibm13	83709	490	99666	357075	224	Peko13	83709	1260	99666	357075	290
ibm14	147088	517	152772	546816	305	Peko14	147088	1672	152772	546816	385
ibm15	161187	383	186608	715823	303	Peko15	161187	1748	186608	715823	402
ibm16	182980	504	190048	778823	347	Peko16	182980	1864	190048	778823	429
ibm17	184752	743	189581	860036	379	Peko17	184752	1872	189581	860036	431
ibm18	210341	272	201920	819697	361	Peko18	210341	1998	201920	819697	460

derived set have a whitespace of 10%. The IBM-Place Benchmarks used in *Dragon* [15] cannot be used because they do not have any connectivity information between the movable cells and the fixed pads, present on the placement boundary. This information is essential for a quadratic placement approach. These modified benchmarks are now available online at [30]. Statistics for the placement benchmarks are given in Table I.

In our experiments we have compared *FastPlace* with state-of-the-art academic placers - *Capo 8.8* [16], *Dragon 2.2.3* [15] and *Gordian-Domino* [10], [31].

B. Comparison between Net Models

To determine the effect of the Hybrid net model on the number of entries in matrix Q and the runtime, we consider two implementations of *FastPlace*: (a) incorporating the clique model (b) incorporating the Hybrid net model. Table II gives the results for the two implementations. It can be seen that on average, the Hybrid model leads to 2.95× fewer non-zero entries in matrix Q as compared to the clique model over the 18 IBM benchmarks. Also, on average, the total runtime of the placer is 1.5× less for the Hybrid net model.

TABLE II
CLIQUE NET MODEL VS HYBRID NET MODEL.

Circuit	#Non-zero Entries		Ratio *(#Clique/ #Hybrid)*	Runtime *(Clique/ Hybrid)*
	(Clique)	(Hybrid)		
ibm01	109183	41164	2.65	1.5
ibm02	343409	70014	4.90	2.4
ibm03	206069	74680	2.76	1.4
ibm04	220423	84556	2.61	1.2
ibm05	349676	108282	3.23	1.3
ibm06	321308	106835	3.01	1.6
ibm07	373328	147009	2.54	1.3
ibm08	732550	173541	4.22	2.0
ibm09	478777	185102	2.59	1.4
ibm10	707969	251101	2.82	1.6
ibm11	508442	230865	2.20	1.2
ibm12	748371	270849	2.76	1.6
ibm13	744500	295048	2.52	1.5
ibm14	1125147	456474	2.46	1.3
ibm15	1751474	607289	2.88	1.4
ibm16	1923995	668491	2.88	1.3
ibm17	2235716	753507	2.97	1.4
ibm18	2221860	711702	3.12	1.4
Avg			**2.95**	**1.5**

C. Runtime Analysis of the Algorithm

Table III gives the total number of global placement iterations and a break-up of the total runtime of *FastPlace*. The table shows the results for two flows (a) incorporating the Iterative Local Refinement (b) without the Iterative Local Refinement. Also, Table IV summarizes the total runtime and final wirelength results for both the flows. It can be seen from Column 2 of Table III that within 31 iterations of global placement (required for ibm16) the algorithm converges to a solution for all benchmark circuits. This demonstrates the effectiveness of the Cell Shifting and Iterative Local Refinement techniques to accelerate the convergence of the conjugate gradient solver so as to obtain a fast global placement solution. From Table III for flow (a) it can be seen that on average the Cell Shifting and Refinement techniques account for 9.9% and 46.1% of the total runtime. Even though, the refinement technique takes up 46.1% of the total runtime, the average speed-up obtained for flow (b) as seen from Table IV is 32.3%. This shows that the Refinement technique also aids in the convergence of the algorithm and hence, in its absence, the other steps of the algorithm take up more time to compensate for it.

D. Comparison Between Placement Tools

The comparison results between *Capo*, *Dragon* and *FastPlace* are generated on a Sun Sparc-2, 750 MHz machine. The results between *Gordian-Domino* and *FastPlace* are generated on a Intel Xeon, 3.06 GHz machine. We run *MetaPl-Capo8.8* for Solaris, which incorporates Capo, orientation optimizer and row ironing, in the default mode. *Dragon* is run in the default mode, *Gordian* is run in the best mode and *Domino* is run in the default mode.

The half-perimeter wirelength (HPWL) and runtime results for *Capo*, *Dragon* and *FastPlace* are given in Table V. For the IBM benchmarks, on average, *FastPlace* is 13.4× faster than *Capo* with the average wirelength being 1.7% less than *Capo*. On average, *FastPlace* is 102× faster than *Dragon* with the average wirelength being 6.9% more. *Dragon* was also ran in the fixed-die mode and the average wirelength of *FastPlace* was actually 2.3% less than *Dragon* for this case. For the PEKO benchmarks, on average, *FastPlace* is 7.6× faster than *Capo* with the average wirelength being 6.6% more. On average, *FastPlace* is 71.5× faster than *Dragon* with the average wirelength being 4.7% less than *Dragon*.

TABLE III
GLOBAL PLACEMENT ITERATIONS AND BREAK-UP OF TOTAL RUNTIME.

Ckt	With Iterative Local Refinement							Without Iterative Local Refinement					
	GP. Iters.	Global Opt.	Cell Shifting	ILR	Det. Place	Total (sec)	HPWL (×10e6)	GP. Iters.	Global Opt.	Cell Shifting	Det. Place	Total (sec)	HPWL (×10e6)
ibm01	19	3.73	1.42	6.83	1.17	13.15	1.89	19	4.06	1.43	1.62	7.11	2.10
ibm02	22	7.93	2.81	15.49	2.90	29.13	3.90	24	8.54	3.28	4.61	16.43	4.40
ibm03	21	9.24	3.23	17.94	2.00	32.41	5.25	22	10.07	3.54	3.73	17.34	6.06
ibm04	21	10.1	4.06	17.81	2.54	34.51	6.22	26	11.22	6.30	5.31	22.83	7.10
ibm05	24	9.18	5.74	26.53	6.61	48.06	10.72	24	10.4	5.7	9.61	25.71	11.94
ibm06	21	14.77	4.43	23.57	3.13	45.90	5.44	21	16.4	4.41	7.84	28.65	6.04
ibm07	24	30.78	8.39	29.24	5.08	73.49	9.01	27	33.77	10.46	8.87	53.1	10.17
ibm08	23	29.26	8.91	43.64	5.01	86.82	9.78	26	32.88	11.34	14.24	58.46	11.12
ibm09	29	44.45	13.90	36.72	7.30	102.37	10.84	30	46.37	14.72	11.06	72.15	12.35
ibm10	22	64.85	11.78	62.69	10.07	149.39	18.89	27	66.68	16.88	17.18	100.74	21.87
ibm11	25	61.83	15.21	50.05	7.15	134.24	15.54	28	62.68	17.89	15.70	96.27	17.56
ibm12	25	66.31	14.42	73.58	6.27	160.58	24.48	31	74.67	21.30	18.07	114.05	27.79
ibm13	24	77.26	15.74	71.12	11.66	175.78	19.08	28	85.15	20.99	22.97	129.11	22.02
ibm14	25	159.86	31.12	118.51	15.74	325.23	35.67	30	241.87	41.85	36.46	320.18	42.56
ibm15	31	221.53	47.82	258.74	21.12	549.21	43.99	37	240.18	63.12	50.76	354.06	52.59
ibm16	31	250.32	56.50	236.10	34.08	577.00	46.59	34	261.55	65.60	71.41	398.56	54.01
ibm17	26	261.87	41.91	260.00	38.24	602.02	67.66	41	316.33	87.00	96.13	499.46	81.41
ibm18	25	265.08	48.70	401.93	54.57	770.29	46.39	33	311.84	79.58	95.97	487.39	56.91

TABLE IV
COMPARISON BETWEEN THE FLOWS WITH AND WITHOUT ITERATIVE LOCAL REFINEMENT.

Ckt	Total Runtime (sec)			HPWL (×10e6)		
	With ILR	Without ILR	% dec	With ILR	Without ILR	% inc
ibm01	13.15	7.11	45.9	1.89	2.10	11.1
ibm02	29.13	16.43	43.6	3.90	4.40	12.8
ibm03	32.41	17.34	46.5	5.25	6.06	15.4
ibm04	34.51	22.83	33.9	6.22	7.10	14.2
ibm05	48.06	25.71	46.5	10.72	11.94	11.4
ibm06	45.90	28.65	37.6	5.44	6.04	11.0
ibm07	73.49	53.1	27.8	9.01	10.17	12.9
ibm08	86.82	58.46	32.7	9.78	11.12	13.7
ibm09	102.37	72.15	29.5	10.84	12.35	13.9
ibm10	149.39	100.74	32.6	18.89	21.87	15.8
ibm11	134.24	96.27	28.3	15.54	17.56	13.0
ibm12	160.58	114.05	29.0	24.48	27.79	13.5
ibm13	175.78	129.11	26.5	19.08	22.02	15.4
ibm14	325.23	320.18	1.55	35.67	42.56	19.3
ibm15	549.21	354.06	35.5	43.99	52.59	19.5
ibm16	577.00	398.56	30.9	46.59	54.01	15.9
ibm17	602.02	499.46	17.0	67.66	81.41	20.3
ibm18	770.29	487.39	36.7	46.39	56.91	22.7
Average			**32.3**			**15.1**

Table VI gives the half-perimeter wirelength and runtime results for *Gordian-Domino* and *FastPlace*. Also, included are results when *FastPlace* was run for global placement and *Domino* was run for detailed placement. For the IBM benchmarks, on average, *FastPlace* is 19.9× faster than *Gordian-Domino* with the average wirelength being just 1.0% more. For the PEKO benchmarks, on average, *FastPlace* is 5.0× faster than *Gordian-Domino* with the average wirelength being 7.2% more.

We believe that *FastPlace* generates a very good global placement solution. Our detailed placement technique on the other hand is a fast, greedy legalizer that needs further improvement. To illustrate the quality of the global placement solution generated by *FastPlace*, we run the *Domino* detailed placer on *FastPlace* global placements. The results are summarized below:

First, compared to *FastPlace*, the *FastPlace-Domino* flow achieves an average reduction of 5.9% and 14.8% (column 6 of Table VI) in the final half-perimeter wirelength for the IBM and PEKO suites respectively.

Second, excluding Peko03, *FastPlace-Domino* generates better results than *Gordian-Domino* for every other benchmark of the IBM and PEKO suites. On average, the *FastPlace-Domino* wirelength is 4.7% and 6.4% less than *Gordian-Domino* for the IBM and PEKO suites respectively. The corresponding speed-up obtained is 2.7× and 1.4× respectively.

Third, *Domino* takes less runtime on *FastPlace* global placements as compared to *Gordian* global placements. It achieves an average speed-up of 1.3× and 1.1× on *FastPlace* placements as compared to *Gordian* placements for the IBM and PEKO suites respectively.

E. Scalability Analysis of the Algorithms

The total number of pins in a circuit is a good measure of the circuit size. To determine the scalability factor of *FastPlace*, we plot the runtime of the algorithm vs the total number of pins, in logarithmic scale for all 18 benchmarks of the IBM suite in Figure 8. The data points can be closely approximated by a straight line with slope 1.38. Hence, the runtime of *FastPlace* is roughly $O(n^{1.38})$, where n is the circuit size given by the number of pins. Based on the above procedure the runtime of the other placement algorithms are approximately: *Capo* - $O(n^{1.15})$, *Dragon* - $O(n^{1.34})$, *Gordian-Domino* - $O(n^{1.29})$. *Capo* happens to be the fastest among the other three placers. Also, the scalability factor of *Capo* is better than that of *FastPlace*. For the circuits tested, *FastPlace* is faster than *Capo*. Using the scalability data for *FastPlace* and *Capo* we determine that the runtime of *FastPlace* will be equal to that of *Capo* when the circuit size is approximately 17 billion pins.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we propose *FastPlace*, an efficient and scalable flat placement algorithm for large-scale standard cell circuits. *FastPlace* is based on the analytical placement approach and utilizes the quadratic wirelength objective. The current implementation handles the wirelength minimization problem. It produces comparable placement solutions to state-of-the-art academic placers, but in a significantly lesser runtime. Such an ultra-fast placement tool is very much needed for the timing convergence of the layout phase of IC design.

TABLE V
COMPARSION OF PLACEMENT RESULTS WITH CAPO 8.8 AND DRAGON 2.2.3.

Ckt	HPWL (×10e6)			HPWL Ratio		RunTime			Speed-up	
	Capo	Dragon	FastPlace	$\frac{FastPlace}{Capo}$	$\frac{FastPlace}{Dragon}$	Capo	Dragon	FastPlace	$\frac{Capo}{FastPlace}$	$\frac{Dragon}{FastPlace}$
ibm01	1.86	1.75	1.89	1.02	1.08	3m 59s	29m 08s	13s	×18.4	×134.5
ibm02	4.06	3.68	3.90	0.96	1.06	7m 15s	32m 26s	29s	×15.0	×67.1
ibm03	5.11	4.81	5.25	1.03	1.09	8m 23s	33m 49s	32s	×15.7	×63.4
ibm04	6.39	5.79	6.22	0.97	1.07	10m 46s	1h 10m	35s	×18.5	×120.0
ibm05	10.56	9.84	10.72	1.01	1.09	10m 44s	1h 53m	48s	×13.4	×141.2
ibm06	5.50	5.04	5.44	0.99	1.08	12m 08s	1h 21m	46s	×15.8	×105.7
ibm07	9.63	8.60	9.01	0.94	1.05	18m 32s	1h 39m	1m 13s	×15.2	×81.4
ibm08	10.26	9.25	9.78	0.95	1.06	19m 53s	4h 32m	1m 27s	×13.7	×187.6
ibm09	10.56	9.92	10.84	1.03	1.09	22m 50s	3h 51m	1m 42s	×13.4	×135.9
ibm10	19.70	18.10	18.89	0.96	1.04	29m 04s	3h 29m	2m 29s	×11.7	×84.2
ibm11	15.73	14.40	15.54	0.99	1.08	31m 11s	2h 21m	2m 14s	×14.0	×63.1
ibm12	25.83	23.36	24.48	0.95	1.05	30m 41s	3h 43m	2m 41s	×11.4	×83.1
ibm13	18.73	17.76	19.08	1.02	1.07	39m 27s	3h 07m	2m 56s	×13.5	×63.8
ibm14	36.69	33.20	35.67	0.97	1.07	1h 12m	7h 58m	5m 25s	×13.3	×88.2
ibm15	43.85	40.10	43.99	1.00	1.10	1h 30m	10h 21m	9m 09s	×9.8	×67.9
ibm16	49.63	44.22	46.59	0.94	1.05	1h 31m	12h 17m	9m 37s	×9.5	×76.6
ibm17	69.07	65.36	67.66	0.98	1.04	1h 43m	27h 05m	10m 02s	×10.3	×162.0
ibm18	47.46	43.42	46.39	0.98	1.07	1h 44m	23h 35m	12m 50s	×8.1	×110.3
Average				**0.983**	**1.069**				**×13.4**	**×102.0**
Peko01	1.47	1.63	1.53	1.04	0.94	2m 25s	18m 31s	13s	×11.2	×85.5
Peko02	2.28	2.45	2.32	1.02	0.95	3m 55s	31m 46s	40s	×5.9	×47.7
Peko03	2.69	2.98	2.91	1.08	0.98	4m 55s	36m 21s	36s	×8.2	×60.6
Peko04	3.32	4.30	3.29	0.99	0.77	6m 09s	1h 10m	32s	×11.5	×131.2
Peko05	3.60	4.03	3.90	1.08	0.97	6m 36s	1h 46m	47s	×8.4	×135.3
Peko06	4.00	4.09	3.91	0.98	0.96	7m 24s	1h 27m	47s	×9.4	×111.1
Peko07	5.11	6.36	5.67	1.11	0.89	10m 44s	1h 02m	1m 22s	×7.9	×45.4
Peko08	5.77	6.01	6.08	1.05	1.01	12m 08s	2h 39m	1m 40s	×7.3	×95.4
Peko09	6.66	8.22	6.77	1.02	0.82	13m 27s	2h 13m	1m 25s	×9.5	×93.9
Peko10	8.95	9.05	9.81	1.10	1.08	18m 35s	3h 00m	2m 34s	×7.2	×70.1
Peko11	8.78	8.97	9.35	1.06	1.04	18m 14s	2h 19m	2m 19s	×7.9	×60.0
Peko12	9.52	9.56	9.70	1.02	1.01	19m 25s	3h 12m	2m 24s	×8.1	×80.0
Peko13	11.23	12.44	12.73	1.13	1.02	23m 23s	3h 02m	2m 57s	×7.9	×61.7
Peko14	17.06	18.26	20.55	1.20	1.12	44m 14s	4h 03m	7m 54s	×5.6	×30.8
Peko15	22.68	25.10	22.86	1.01	0.91	53m 18s	5h 33m	8m 17s	×6.4	×40.2
Peko16	24.59	29.18	26.29	1.07	0.90	1h 02m	6h 07m	11m 55s	×5.2	×30.8
Peko17	26.13	33.16	27.79	1.06	0.84	1h 04m	12h 28m	11m 56s	×5.4	×62.7
Peko18	25.04	31.01	29.32	1.17	0.95	1h 11m	11h 18m	15m 13s	×4.7	×44.6
Average				**1.066**	**0.953**				**×7.6**	**×71.5**

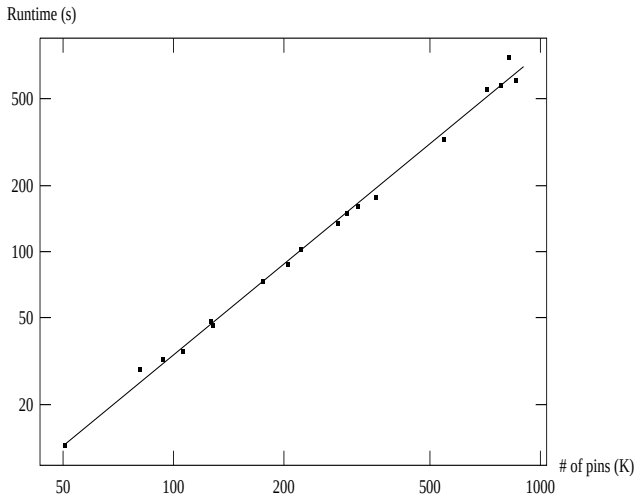

Fig. 8. Runtime of FastPlace vs number of pins in logarithmic scale.

The runtime of *FastPlace* can be further reduced by: (a) Employing a hierarchical framework (e.g. [19]) to reduce the problem size. The reduced problem can then be solved by *FastPlace*. We show empirically that the time complexity of *FastPlace* is roughly $O(n^{1.38})$. Hence, if the circuit size is reduced by half, the runtime of *FastPlace* can be reduced by a factor of 2.6. (b) By using the algebraic multigrid method [32] to solve the system of linear equations (5).

The *FastPlace* algorithm can also be extended to consider other placement objectives like mixed-mode placement, timing driven placement, routability driven placement, variable whitespace allocation, etc. Future extensions to the algorithm would be in dealing with the above objectives.

ACKNOWLEDGMENT

The authors would like to thank Prof. Frank Johannes of the Technical University of Munich for access to the *Plato/Domino* package and Bernd Obermeier for answering their queries regarding the same. They would like to thank, Prof. Jason Cong, Joe Shinnerl and Min Xie of UCLA for providing the PEKO benchmarks with pads. They would also like to thank Xiaojian Yang and Saurabh Adya, currently with Synplicity, Inc. for discussions regarding the IBM benchmarks and the placement tools *Dragon* and *Capo* respectively.

TABLE VI
COMPARSION OF PLACEMENT RESULTS WITH GORDIAN-DOMINO AND FASTPLACE-DOMINO.

Ckt	HPWL (×10e6)			HPWL Ratio		RunTime			Speed-up	
	Gordian Domino	FastPlace Domino	FastPlace	$\frac{FP}{GN+DO}$	$\frac{FP}{FP+DO}$	Gordian Domino	FastPlace Domino	FastPlace	$\frac{GN+DO}{FP}$	$\frac{FP+DO}{FP}$
ibm01	1.85	1.70	1.89	1.02	1.11	2m 04s	44s	5s	×24.8	×8.8
ibm02	3.94	3.69	3.93	1.00	1.07	5m 07s	1m 56s	15s	×20.5	×7.7
ibm03	5.13	4.95	5.27	1.03	1.06	5m 24s	1m 36s	14s	×23.1	×6.9
ibm04	6.31	5.79	6.15	0.97	1.06	5m 56s	1m 59s	15s	×23.7	×7.9
ibm05	10.51	10.30	10.59	1.00	1.03	10m 02s	2m 28s	19s	×31.7	×7.8
ibm06	5.17	5.04	5.41	1.05	1.07	9m 08s	5m 01s	22s	×24.9	×13.7
ibm07	9.33	8.64	9.10	0.98	1.05	13m 40s	5m 29s	39s	×21.0	×8.4
ibm08	9.79	9.34	9.80	1.00	1.05	21m 19s	6m 03s	52s	×24.6	×7.0
ibm09	10.34	10.17	10.79	1.04	1.06	14m 27s	6m 57s	46s	×18.9	×9.1
ibm10	19.41	18.13	18.97	0.98	1.05	27m 06s	10m 37s	1m 30s	×18.1	×7.1
ibm11	15.57	14.56	15.52	1.00	1.07	21m 12s	11m 04s	1m 18s	×16.3	×8.5
ibm12	23.72	23.51	24.56	1.04	1.04	28m 29s	11m 46s	1m 28s	×19.4	×8.0
ibm13	18.44	17.71	18.92	1.03	1.07	29m 12s	13m 16s	1m 54s	×15.4	×7.0
ibm14	36.27	33.91	35.68	0.98	1.05	52m 50s	25m 24s	3m 22s	×15.7	×7.5
ibm15	42.97	42.16	44.38	1.03	1.05	1h 07m	27m 24s	5m 21s	×12.5	×5.1
ibm16	47.88	44.25	46.93	0.98	1.06	1h 17m	32m 20s	5m 06s	×15.1	×6.3
ibm17	65.96	64.35	67.44	1.02	1.05	1h 35m	40m 33s	5m 59s	×15.9	×6.8
ibm18	44.61	43.63	46.26	1.04	1.06	1h 50m	32m 09s	6m 31s	×16.9	×4.9
Average				**1.010**	**1.059**				**×19.9**	**×7.7**
Peko01	1.29	1.28	1.53	1.19	1.20	40s	23s	6s	×6.7	×3.8
Peko02	2.03	2.03	2.31	1.14	1.14	1m 21s	1m 03s	16s	×5.1	×3.9
Peko03	2.59	2.61	2.86	1.10	1.10	1m 27s	59s	14s	×6.2	×4.2
Peko04	3.07	3.01	3.28	1.07	1.09	1m 46s	1m 08s	17s	×6.2	×4.0
Peko05	3.32	3.16	3.95	1.19	1.25	2m 04s	1m 32s	22s	×5.6	×4.2
Peko06	3.75	3.65	3.86	1.03	1.06	2m 08s	1m 36s	23s	×5.6	×4.2
Peko07	5.57	5.43	5.70	1.02	1.05	3m 23s	2m 36s	43s	×4.7	×3.6
Peko08	6.03	5.81	6.04	1.00	1.04	5m 17s	3m 46s	1m 02s	×5.1	×3.7
Peko09	7.13	6.54	6.88	0.96	1.05	4m 13s	4m 06s	57s	×4.4	×4.3
Peko10	9.10	8.34	9.72	1.07	1.17	7m 06s	5m 30s	1m 30s	×4.7	×3.7
Peko11	9.28	8.64	9.30	1.00	1.08	7m 27s	5m 51s	1m 18s	×5.7	×4.5
Peko12	9.35	8.51	9.99	1.07	1.17	7m 13s	5m 11s	1m 22s	×5.3	×3.8
Peko13	11.72	10.53	11.68	1.00	1.11	8m 48s	7m 02s	1m 53s	×4.7	×3.7
Peko14	18.15	17.85	20.28	1.12	1.14	16m 15s	12m 46s	4m 31s	×3.6	×2.8
Peko15	22.81	18.91	22.98	1.01	1.22	20m 57s	16m 31s	5m 23s	×3.9	×3.1
Peko16	24.19	20.79	26.53	1.10	1.28	26m 58s	17m 27s	6m 33s	×4.1	×2.7
Peko17	26.05	21.72	28.53	1.10	1.31	32m 31s	18m 22s	7m 22s	×4.4	×2.5
Peko18	26.07	24.19	29.38	1.13	1.21	31m 16s	20m 07s	8m 37s	×3.6	×2.3
Average				**1.072**	**1.148**				**×5.0**	**×3.7**

REFERENCES

[1] R. Varadarajan, "Convergence of placement technology in physical synthesis: Is placement really a point tool?" in *Proc. Intl. Symp. on Physical Design*, 2003, p. 7.

[2] P. Villarrubia, "Important placement considerations for modern VLSI chips," in *Proc. Intl. Symp. on Physical Design*, 2003, p. 6.

[3] *The International Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 2000.

[4] C.-C. Chang, J. Cong, and M. Xie, "Optimality and scalability study of existing placement algorithms," in *Proc. Asian and South Pacific Design Automation Conf.*, 2003, pp. 621–627.

[5] J. Cong, M. Romesis, and M. Xie, "Optimality, scalability, stability study of partitioning and placement algorithms," in *Proc. Intl. Symp. on Physical Design*, 2003, pp. 88–94.

[6] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytical placer," in *Proc. Intl. Symp. on Physical Design*, 2004, pp. 18–25.

[7] H. Eisenmann and F. Johannes, "Generic global placement and floorplanning," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 269–274.

[8] H. Etawil, S. Arebi, and A. Vannelli, "Attractor-repeller approach for global placement," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 1999, pp. 20–24.

[9] B. Hu and M. Marek-Sadowska, "FAR: Fixed-points addition and relaxation based placement," in *Proc. Intl. Symp. on Physical Design*, 2002, pp. 161–166.

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

[11] G. Sigl, K. Doll, and F. Johannes, "Analytical Placement: A linear or a quadratic objective function," in *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 427–431.

[12] S.-W. Hur, T. Cao, K. Rajagopal, Y. Parasuram, A. Chowdhary, V. Tiourin, and B. Halpin, "Force directed mongrel with physical net constraints," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 214–219.

[13] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 746–751.

[14] C. Sechen and A. L. Sangiovanni-Vincentelli, "TimberWolf 3.2: A new standard cell placement and global routing package," in *Proc. ACM/IEEE Design Automation Conf.*, 1986, pp. 432–439.

[15] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2000, pp. 260–263.

[16] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection produce routable placements," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 477–482.

[17] T. Chan, J. Cong, T. Kong, and J. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2000, pp. 171–176.

[18] M. C. Yildiz and P. H. Madden, "Global objectives for standard cell placement," in *Proc. 11th Great Lakes Symposium on VLSI*, 2001, pp. 68–72.

[19] B. Hu and M. Marek-Sadowska, "Fine granularity clustering for large

scale placement problems," in *Proc. Intl. Symp. on Physical Design*, 2003, pp. 67–74.
[20] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[21] S.-W. Hur and J. Lillis, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2000, pp. 165–170.
[22] W. Naylor *et al.*, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," U.S. Patent 6 301 693, Oct., 2001.
[23] F. Mo, A. Tabbara, and R. Brayton, "A force-directed macro-cell placer," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2000, pp. 177–180.
[24] K. M. Hall, "An r-dimensional quadratic placement algorithm," *Management Science*, vol. 17, pp. 219–229, 1970.
[25] D. S. Kershaw, "The Incomplete Cholesky-Conjugate Gradient method for the iterative solution of systems of linear equations," *J. Comp. Physics*, vol. 26, pp. 43–65, 1978.
[26] R. Barrett *et al.*, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. SIAM, 1994.
[27] S. N. Adya and I. L. Markov. ISPD02 IBM-MS Mixed-size Placement Benchmarks. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/ISPD02bench/
[28] ——, "Consistent placement of macro-blocks using floorplanning and standard-cell placement," in *Proc. Intl. Symp. on Physical Design*, 2002, pp. 12–17.
[29] J. Cong, C.-C. Chang, and M. Xie. PEKO Placement Benchmark Suite. [Online]. Available: http://ballade.cs.ucla.edu/~pubbench/
[30] N. Viswanathan and C. C.-N. Chu. ISPD04 IBM Standard Cell Benchmarks with Pads. [Online]. Available: http://www.public.iastate.edu/~nataraj/ISPD04_Bench.html
[31] K. Doll, F. M. Johannes, and K. J. Antreich, "Iterative placement improvement by network flow methods," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 10, pp. 1189–1200, Oct. 1994.
[32] H. Chen, C.-K. Cheng, N.-C. Chou, A. Kahng, J. MacDonald, P. Suaris, B. Yao, and Z. Zhu, "An algebraic multigrid solver for analytical placement with layout based clustering," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 794–799.

Natarajan Viswanathan received the B.E. degree in electronics and communication engineering from Karnataka Regional Engineering College, Surathkal, India, in 2001 and the M.S. degree in computer engineering from Iowa State University, in 2003. He is currently working towards the Ph.D. degree in computer engineering at Iowa State University.

His research interests are in VLSI physical design, specifically in algorithms and methodologies for placement.

Chris C. N. Chu received the B.S. degree in computer science from the University of Hong Kong, Hong Kong, in 1993. He received the M.S. degree and the Ph.D. degree in computer science from the University of Texas at Austin in 1994 and 1999, respectively.

Dr. Chu is currently an Assistant Professor in the Electrical and Computer Engineering Department at Iowa State University. His research interests include design and analysis of algorithms, CAD of VLSI physical design, and performance-driven interconnect optimization. He received the IEEE TCAD best paper award at 1999 for his work in performance-driven interconnect optimization. He received the ISPD best paper award at 2004 for his work in efficient placement algorithm. He received the Bert Kay Best Dissertation Award for 1998-1999 from the Department of Computer Sciences in the University of Texas at Austin.

Dr. Chu has served on the technical program committees of several major conferences including ISPD, ISCAS, DATE, and ASP-DAC. He has also served as an organizer for the ACM SIGDA Ph.D. Forum.