

# Multilevel Optimization for Large-Scale Circuit Placement \*

Tony F. Chan,<sup>†</sup> Jason Cong,<sup>‡</sup> Tianming Kong,<sup>‡</sup> Joseph R. Shinnerl<sup>‡</sup>

## Abstract

We have designed and implemented a new class of fast and highly scalable placement algorithms that directly handle complex constraints and achieve total wirelengths comparable to the state of the art. Our approach exploits recent advances in (i) multilevel methods for hierarchical computation, (ii) interior-point methods for nonconvex nonlinear programming, and (iii) the Fast Multipole Method for the order  $N$  evaluation of sums over the  $N(N-1)/2$  pairwise interactions of  $N$  components. Significant adaptation of these methods for the placement problem is required, and we have therefore developed a set of customized discrete algorithms for clustering, declustering, slot assignment, and local refinement with which the continuous algorithms are naturally combined. Preliminary test runs on benchmark circuits with up to 184,000 cells produce total wirelengths within approximately 5–10% of those of GORDIAN-L [1] in less than one tenth the run time. Such an ultra-fast placement engine is badly needed for timing convergence of the synthesis and layout phases of integrated circuit design.

## 1 Introduction

In nanometer IC technologies, existing placement methods face two serious challenges: large design sizes (over one million placeable objects) and complex design constraints (delay, noise, manufacturability, etc.). The well-known simulated annealing method (SA) [2, 3] can handle complex design constraints. But since it requires a slow annealing process and moves only small numbers of cells at each step, its runtime and quality scale poorly as the design size increases. Commonly used quadratic-programming (QP) based placement techniques [4] are very efficient, but they usually assume linear constraints and cannot handle complex constraints. For example, since pairwise nonoverlap cell constraints are linearized in the QP formulation, QP-based techniques resort to a sequence of successive partitioning and quadratic placement iterations to get nonoverlap solutions. Such processes practically ignore timing constraints in early iterations, as, due to cell overlap, timing constraints are trivially satisfied in early iterations.

There are two general objectives for circuit placement research. One is to improve the solution quality in terms of wirelength, circuit performance, etc.. The other is to reduce the runtime and improve the scalability of the algorithm. This paper focuses on the latter. We have implemented a highly efficient and scalable placement engine with solution quality comparable to the state of the art. Scalability is important for two reasons. First, circuit sizes have increased exponentially with Moore's Law. The placement runtime is now a bottleneck in the design process. Second, as the placement result largely determines the interconnect performance, high-level and logic-level

synthesis procedures may need to compute placement solutions repeatedly to measure the synthesis results. An ultra-fast placement engine is badly needed for timing convergence of synthesis and layout.

Some recent work on circuit placement by others uses some ideas similar to ours. Sankar and Rose [5] have implemented a discrete multilevel scheme that obtains large-scale placements rapidly at a 10-30% increase in wirelength compared to their own SA-based placement engine. Hierarchical SA-based placement has been implemented by Sarrafzadeh and Wang [6] and is also used in the TimberWolf package [3]. Eisenmann and Johannes [7] employ a fast convolution to approximate solutions to a Poisson equation for cell density and use these to update a quadratic model. Their method obtains placements of high quality but may not be scalable enough for practical use on circuits with over 100,000 cells.

Our method is the first that successfully integrates constrained nonconvex nonlinear programming into a multilevel framework for circuit placement. Direct handling of the order  $N^2$  nonoverlap constraints, a challenge not accepted by previous authors, is crucial to this integration. At coarser levels of refinement, cluster dimensions may vary widely; the ability to model this variation accurately at these coarse levels significantly improves the placement quality.

Our approach also supports the direct inclusion of complex design constraints such as timing, noise, and manufacturability considerations, without necessarily requiring analytical closed-form representations. It requires only that computational procedures exist to evaluate the constraints and their derivatives for each given placement configuration. However, since the existing benchmarks in the public domain do not have sufficient information for delay and noise calculation, in this paper, we consider only nonoverlap constraints. We currently seek cooperation with SRC member companies to apply our techniques to state-of-the-art placement problems with delay and noise constraints. Our current objective function is the squared wirelength used in GORDIAN and PROUD [4, 8], but our formulation is not restricted to this model.

## 2 Problem Formulation

Given connectivity specifications for a large assembly of rectangular circuit modules, we wish to determine an arrangement of the modules aligned along the rows of a rectangle in the plane so that total wirelength is minimized subject to (i) the connectivity requirements (ii) the requirement that no two modules overlap spatially, and (iii) any other constraints involving interference, power dissipation, routability, etc.. Mathematically, the placement problem can be expressed as a special class of *nonlinear integer programming problem*,

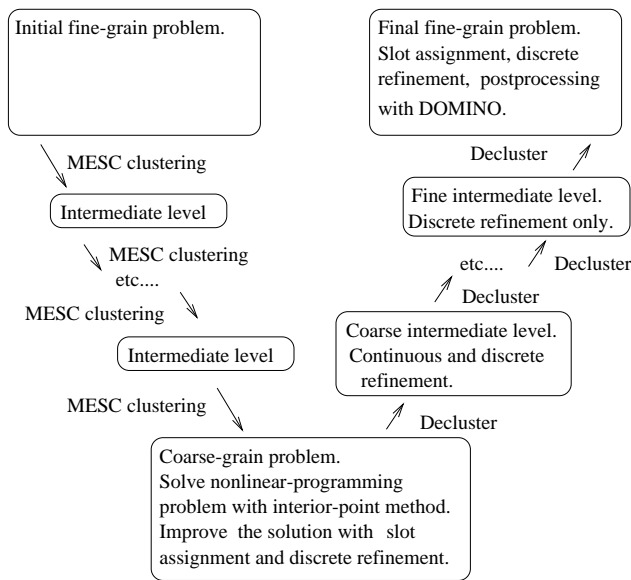
$$\begin{aligned} & \min_{x \in \mathbb{R}^n} && f(x) \\ \text{subject to} &&& c_i(x) \geq 0, \quad i = 1, \dots, m \\ &&& x \in \mathcal{D} \subset \mathbb{R}^n, \end{aligned} \quad (\text{NIP})$$

\*The authors gratefully acknowledge SRC award 99-TJ-686, NSF grant CCR-9901153, and a gift from Intel Corporation.

<sup>†</sup>Mathematics Department, UCLA; Los Angeles, CA 90095-1555; chan@math.ucla.edu

<sup>‡</sup>Computer Science Department, UCLA; Los Angeles, CA 90095-1596; {cong, kongtm, shinnerl}@cs.ucla.edu, Tel: (310)206-2775

Figure 1: Multilevel Placement V-Cycle



where the *objective*  $f$  and the constraints  $c_i$  are scalar-valued functions defined on  $\mathcal{D}$ . Set  $\mathcal{D}$  is disconnected because cells must ultimately be aligned in rows. Relaxing this requirement, however, is a useful simplification that produces a *nonlinear programming problem* (NP) when  $f$  and the  $c_i$  are twice continuously differentiable in the *feasible region*  $\mathbb{F} = \{x \mid c(x) \geq 0\}$ .

Nonconvexity of the feasible region implies that, even if  $f$  is convex, NP may have many different local minima. Although practical algorithms for NP can guarantee convergence only to local 2nd-order minimizers, they use sequences of *global* refinements (i.e., simultaneous displacements of all cells) to obtain high-quality approximations to global minimizers. On small circuits (less than 1000 cells), our nonlinear-programming engine generates placements that are typically comparable to or better than those generated by GORDIAN-L, with occasional improvement up to 20%.

For a placement problem with  $N$  cells, there are  $N(N-1)/2$  pairwise nonoverlap constraints — too many to be efficiently represented or evaluated explicitly, even at the relatively coarse levels of refinement at which our nonlinear-programming engine is applied. To overcome this obstacle, we have implemented an adaptation of the Fast Multipole Method for the efficient  $\mathcal{O}(N)$  evaluation of sums over all  $\mathcal{O}(N^2)$  nonoverlap constraints (Section 5 below).

### 3 Multilevel Placement

Multilevel methods have been studied extensively over the last 20 years as a means of accelerating numerical algorithms for partial differential equations [9, 10]. Application areas are quite diverse: image processing, combinatorial optimization, control theory, statistical mechanics, quantum electrodynamics, and linear algebra. Recently, multilevel techniques have also been applied very successfully to circuit partitioning in the hMetis method [11].

Our multilevel framework for placement is illustrated in Figure 1. The original placement problem is called the *fine-grain*

problem. The recursive clustering procedure MESC (Section 7 below) is applied to reduce the fine-grain problem to a *coarse-grain* problem in stages, each of which can be viewed as a scale of resolution or level of abstraction. Recursive coarsening is known in the multilevel literature as a “downward pass.” Once the scale has been reduced to about 1000 cells, the placement problem is represented within a general, systematic, and standard nonconvex nonlinear programming formulation, NP. A customized interior-point method (Section 4 below) is then used to approximate a global minimizer. Slot assignment and discrete refinement are then used to improve the continuous approximation before proceeding. Recursive declustering and refinement are used to transform back to the fine-grain problem in the so called “upward pass.” At *every* stage of the upward pass, a customized discrete method of localized exchanges (Section 6 below) is used to improve the placement at that scale. At relatively coarse levels along the upward pass, a small number of interior-point iterations is also used as a means of continuous refinement of the placement at those levels. Our tests suggest that the combination of localized, discrete optimization with globalized, continuous optimization is most effective in improving placement quality.

Several variations on this basic structure are possible; in experiments to date we have examined only a few (Section 8 below), and these, only briefly. Our declustering strategy is a simple linear assignment to nearby locations; more sophisticated strategies may improve quality. We have found no noticeable improvement in using both connectivity and spatial information to cluster, even though this strategy allows additional optimization to be performed during the downward pass. Multiple or recursive V-cycles are a common multilevel technique that may improve our method if their properties for the placement problem can be better understood.

### 4 Interior-Point Methods for NP

An interior-point method [12] for the numerical solution of NP can be expressed as the construction of a sequence  $x_k$  of strictly feasible points (i.e.,  $c_i(x_k) > 0$  for all  $i$ ), each of which closely approximates the solution to a barrier subproblem,

$$\min_{\{x \in \mathbb{R}^n \mid c(x) > 0\}} B_\mu(x) \equiv f(x) - \mu \sum_{i=1}^m \ln(c_i(x)). \quad (\text{BP}(\mu))$$

The *barrier parameter*  $\mu$  is a small positive scalar that is fixed for each subproblem. The *barrier function*  $B_\mu(x)$  resembles  $f$  inside the strict interior of the feasible region,  $\{x \mid c_i(x) > 0 \text{ for all } i\}$ , but  $B_\mu(x)$  diverges to positive infinity as  $x$  approaches the boundary of the strict interior from within. Under mild conditions, a sequence  $x(\mu)$  of solutions to  $\text{BP}(\mu)$  will converge to a solution of NP as the barrier parameter  $\mu$  decreases to zero.

The attractive theoretical and practical characteristics of barrier methods have been well documented, especially since the discovery in the mid-80s that there exist efficient implementations of these algorithms with polynomial time complexity for convex problems. Interior-point techniques have recently been used in circuit placement to solve sequences of linear and convex quadratic subproblems [13, 14].

Conceptually, we may think of the barrier transformation as introducing a short-range repulsive force between cells; i.e., we may think of the log-barrier method as an extension of previously attempted force-directed methods for circuit placement [15, 7]. Properly placing the barrier subproblem within the larger nonlinear programming context, however, makes avail-

able a wealth of effective techniques and removes the need for *ad hoc* strategies for selecting parameter values.

An interior-point method is readily extended so as not to require strict satisfaction of the constraints at each step. An artificial variable  $\xi$  and quadratic penalty term can be introduced at negligible cost to allow a gradual reduction in constraint violations. Strict interiority is maintained in the higher-dimensional space that includes both physical and artificial variables. Our implementation uses a single artificial variable and a quadratic penalty as follows. The barrier function is

$$B_\mu(x, \xi) \equiv f(x) - \mu \sum_{i=1}^m \ln(c_i(x) + \xi) + \alpha(\xi + \xi^2),$$

and the associated barrier subproblem is

$$\min_{\{x, \xi\} \in \mathbb{R}^{n+1} | c(x) + \xi > 0} B_\mu(x, \xi). \quad (\text{BP}(\mu, \xi))$$

The scaling parameter  $\alpha > 0$  can be selected for best performance and held fixed. A sufficiently large positive initial value of the artificial variable makes the initial feasible region  $\mathbb{F} = \{x \mid c(x) \geq 0\}$  simply connected. Physically, cells are initially allowed to overlap even to the point of sharing the same center. This initial relaxation of the nonoverlap constraints is necessary to allow the barrier method to move cells freely past one another toward ever-improving configurations. Eventually, the method reduces the artificial variable close enough to zero that only small amounts of overlap are tolerated. The final feasible region thus has order  $N^2$  “holes” corresponding to the explicit prohibition of pairwise overlap. During intermediate iterations, as  $\mu$  decreases and the holes grow in size, cell freedom of movement decreases.

We use a Newton-based linesearch algorithm to solve each barrier subproblem. At each iteration, a search direction  $p$  and a “steplength”  $\alpha$  are computed, in that order. The update to the current iterate is simply  $x \leftarrow x + \alpha p$ . The “steplength”  $\alpha$  is computed using a special algorithm developed by Murray and Wright [16].

Guaranteeing convergence of the subproblem iterations in reasonable time requires careful formulation of the search-direction equations and a special adaptation of the algorithm used to solve them. Our method is based on conjugate gradients with extensions studied by Nash [17]. When sufficient indefiniteness in the barrier function Hessian is detected, the iterations are modified and quickly terminated, producing a direction of both sufficient descent and sufficiently negative curvature to ensure eventual convergence to a local minimizer. Once a neighborhood of a local minimizer is found, the Hessian of the barrier function becomes positive definite, and an approximation to the Newton step is obtained. Early termination of the conjugate gradient steps can be used to make the approximation just accurate enough to ensure superlinear convergence to barrier subproblem solutions. To date our experiments have focused on generic incomplete LU (truncated Gaussian elimination) preconditioners [18] for the Newton equations; these have provided adequate performance gains.

To avoid loss of accuracy due to asymptotic ill conditioning in the Newton equations for the barrier subproblem, the search direction can be decomposed into two well-conditioned orthogonal components, and these can be computed separately. However, if only relatively large values of  $\mu$  are used in  $\text{BP}(\mu, \xi)$ , then such an orthogonal decomposition, which may require an explicit approximation of the constraints and their tangent space, is unnecessary.

The most computationally intensive tasks associated with this log-barrier approach applied to large-scale circuit placement are

(i) evaluation of the objective function, constraints, and their derivatives, and (ii) calculation of the search direction  $p$  at each iteration of the barrier subproblem. Both of these are accelerated by a customized implementation of the Fast Multipole Method.

## 5 The Fast Multipole Method

In circuit placement, the cost of explicitly evaluating all nonoverlap constraints individually at every iteration is prohibitive, making efficient constraint approximation essential. To simultaneously include the effects of all constraints, we rely on the order  $N$  evaluation of order  $N^2$  nonoverlap constraints by the Fast Multipole Method (FMM). Since its discovery in the mid 1980s for the efficient numerical simulation of particle systems, FMM has been applied with great success to such diverse areas as computational chemistry, eigenvalue calculation [19], and capacitance extraction [20]. The improvement of the order  $N$  algorithm over relatively simpler order  $N \log N$  methods is quite significant once  $N$  approaches or exceeds 1000. FMM is also used in our placement engine to evaluate the matrix-vector products in our Krylov-based solver for the search direction. Our application requires a special adaptation of FMM but still yields a significant speed-up over the naive  $N^2$  calculation.

Consider the problem of avoiding overlapping circuit modules. Suppose cell  $i$  is circular with center  $(x_i, y_i)$  and radius  $r_i$ , and similarly for cell  $j$ . For  $i, j = 1, \dots, N$  and  $i > j$ , we have  $N(N-1)/2$  constraints of the form

$$c_{ij}(x) \equiv \|(x_i, y_i) - (x_j, y_j)\|^2 - (r_i + r_j)^2 \geq 0, \quad i > j, \quad (1)$$

where “ $\|\cdot\|$ ” denotes the usual 2-norm distance. The contribution to the barrier function  $B_\mu(x, \xi)$  associated with the  $i$ th constraint is a sum of the form

$$-\mu \sum_{j < i} \ln(c_{ij} + \xi), \quad (2)$$

and we must evaluate  $N-1$  such sums. Each term in the sum (2) above can be viewed a perturbation of the logarithmic potential

$$-\ln \|(x_i, y_i) - (x_j, y_j)\|^2, \quad (3)$$

and for most terms, the perturbation is relatively small. Ignoring the perturbation altogether corresponds to a point-particle approximation of the cells. The Fast Multipole Method reduces the cost of evaluation of  $M$  sums of the form (2) from  $\mathcal{O}(MN)$  to  $\mathcal{O}(M) + \mathcal{O}(N)$ , with speedups of several orders of magnitude when  $N$  is large (over 1000, say).

Some essential elements of FMM are

- (i) clustering particles into boxes and representing the effect of each box in (2) by a Taylor expansion,
- (ii) analytical formulas for translating (recentering) and merging of these “box expansions” to form new expansions,
- (iii) a nested sequence of partitions and a means for obtaining multipole expansions at finer partitions by reusing multipole expansions at coarser partitions, and vice-versa.

In effect, the method imposes hierarchically organized computation on a large collection of elements with no predefined hierarchical structure. A special adaptation is required for our application, due to the presence of the artificial variable,  $\xi$ , and the nonzero cell sizes.

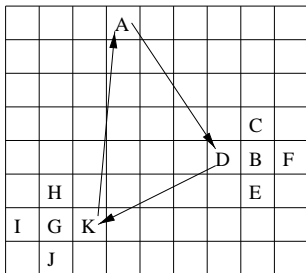
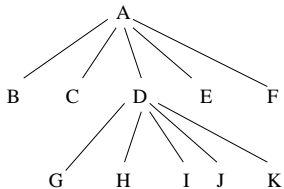
Figure 2:  $\epsilon$ -neighborhood

Figure 3: Search Tree from A



## 6 Discrete Refinement

At all levels of refinement, localized permutations of cells are used to improve total wirelength at very little cost in runtime. First we partition the placement area into a set of regular slots and assign cells into slots by linear assignment. Then we permute small subsets of cells to decrease the objective.

The algorithm is based on the concepts of  $\epsilon$ -neighborhood and  $\lambda$ -exchange [21]. Assuming all cells except  $v$  are fixed in their current locations, we can compute  $v$ 's "optimal" slot locations.\* Suppose  $v$ 's optimal slot location is row  $r$ , column  $c$ . Cells located in slots at row  $i$ , column  $j$  where  $|i - r| + |j - c| \leq \epsilon$  are called  $\epsilon$ -neighbors of cell  $v$ . For example, in Figure 2, suppose the optimal slot location of cell  $A$  is occupied by cell  $B$ .  $A$ 's 1-neighbors are  $\{B, C, D, E, F\}$ . Similarly, assuming that  $D$ 's optimal slot is taken by  $G$ , we say cell  $D$ 's 1-neighbors are<sup>†</sup>  $\{G, H, I, J, K\}$ .

We use a different  $\lambda$ -exchange algorithm from [21]. In [21], starting from a cell  $v_1$ , we compute all of its  $\epsilon$ -neighbors; then for each cell in  $v_1$ 's  $\epsilon$ -neighborhood, compute its  $\epsilon$ -neighbors, and so on. This procedure generates a search tree, each leaf of which defines a cell-exchange sequence. For example, part of the search tree from cell  $A$  is shown in Figure 3. With  $\epsilon = 1, \lambda = 3$ , leaf  $K$  defines the following exchange sequence:  $A \rightarrow D \rightarrow K \rightarrow A$ , i.e., move cell  $A$  to  $D$ 's slot, move  $D$  to  $K$ 's slot, and move  $K$  to  $A$ 's slot. The best cell exchange sequence will be chosen, or no exchange is made, if the original placement has a smaller cost. This method has two major drawbacks: first, the size of the search tree grows very quickly with slight increase of  $\epsilon$  and  $\lambda$ ; second, the cell exchange sequence may not be the best possible. Intuitively, moving a cell into its  $\epsilon$ -neighborhood has a high probability of reducing the objective function value, but the last step, moving cell  $v_\lambda$  to the slot of  $v_1$ , may not be good.

To address these problems, we revise the  $\lambda$ -exchange procedure as follows. Suppose  $v_1$  is the starting cell. We compute

\*Since we are using quadratic wirelength as the objective function, this computation can be done very efficiently.

<sup>†</sup>Our definition is different from [21], where  $\epsilon$  is defined to be the number of child nodes for each node in the search tree. For this example, one would say  $\{B, C, D, E, F\}$  are 5-neighbors of  $A$ .

its  $\epsilon$ -neighbors and randomly pick one cell, say  $v_2$ , from these. Then for  $v_2$ , we compute its  $\epsilon$ -neighbors, and randomly pick one cell, and continue in this fashion until we have  $\lambda$  cells. Then for these  $\lambda$  cells, we try all of their placement permutations (the total number is  $\lambda!$ ) and exchange cells according to the least cost permutation. For example, suppose we pick cells  $A, D$ , and  $K$ . Then all six permutations will be tried: no exchange,  $A \leftrightarrow D$ ,  $A \leftrightarrow K$ ,  $D \leftrightarrow K$ ,  $A \rightarrow D \rightarrow K \rightarrow A$ ,  $A \rightarrow K \rightarrow D \rightarrow A$ . The number of solutions we search still goes exponentially with  $\lambda$ , but not with  $\epsilon$ .

The benefit of randomly selecting  $\epsilon$ -neighbors of the optimal slot is that it supports multiple passes across the placement region. In our current implementation, we use up to 10 passes with  $\epsilon = 1, \lambda = 5$ . Experimentally we found that [21]'s algorithm quickly gets stuck at a local minimizer, while our algorithm has a higher probability of finding better solutions.

## 7 Clustering

We tested three kinds of clustering algorithms: a connectivity based multilevel clustering algorithm using edge separability (MESCG [22]), a hybrid strategy based on both connectivity and relative positions (MESCG), and for comparison, a very simple clustering method based on random matching.

All these clustering algorithms are based on iterative edge matching on a weighted graph  $G$ : each time we pick one edge  $e$  with maximum *rank* and see if clustering the corresponding two nodes satisfies the cluster-size constraint. If so, we cluster (or "match") these two nodes and delete them from the graph; otherwise we discard the edge. The algorithms differ only in their definitions of edge rank.

**MESCG**— Unlike algorithms making use of local connectivity information in the netlist, MESCG uses the more global connectivity notion of "edge separability" to guide the clustering process while maintaining cluster size balance. Computing edge separability for a given edge  $e = (s, t)$  in a graph  $G$  is equivalent to computing the  $s$ - $t$  min-cut, which is quite expensive. It is shown in [22] that a simple and efficient algorithm CAPFOREST [23] can be used to estimate edge separability for all edges of  $G$  in  $\mathcal{O}(n \log n)$  time, where  $n$  is the number of nodes, without flow computation.

In MESCG, the rank of edge  $s = (s, t)$  is defined as:

$$r(e) = \frac{q(e)}{\min(a(s), a(t))},$$

where  $a(s)$  denotes the size of  $s$ . One can see that the definition of  $r(e)$  gives higher priority to edges with larger  $q(e)$  and edges whose contraction leads to a smaller increase of cell size, i.e., we want to cluster nodes with strong connection and also maintain cluster size balance.

**MESCG**— Here it is assumed that before clustering, we have a placement. During the first V-cycle, we can use the initial unconstrained optimal solution; for subsequent V-cycles, we can use the placement from the previous V-cycle.

For each edge  $e = (s, t)$ , we define its rank as follows:

$$r(e) = \frac{q(e)}{(1 + d(s, t)^\alpha) \min(a(s), a(t))}$$

where  $d(s, t)$  denotes the distance between cell  $s$  and  $t$ ,  $\alpha$  is a constant. In our current implementation,  $\alpha$  is set to be 0.3. The motivation for this definition is that we believe that the current placement is "good": two cells currently close to each other are more likely to be close to each other in the final placement.

Table 1: Test Circuits

circuit	# cells	# nets	# pads	# rows
struct	1888	1920	64	21
biomed	6417	5742	97	46
avqsmall	21854	22124	64	80
avqlarge	25114	25384	64	80
ibm04	27220	31970	287	116
ibm07	45639	48117	287	151
ibm09	53110	60902	285	162
ibm14	147088	152772	517	271
ibm17	184752	189581	743	303

Table 2: Impact of Nonlinear Programming

levels	wirelength	CPU
0	4.35e+06	134.2
1	4.04e+06	136.6
2	4.01e+06	127.3
3	3.99e+06	136.9
4	3.91e+06	163.3
5	3.86e+06	336.9

**RandomMatch**— This algorithm is used purely for comparison purposes. Edge ranks are randomly assigned.

## 8 Experimental Results

The results in this section are preliminary. Many variations on the framework discussed in the preceding sections are possible; our experiments to date have not explored them all.

Table 1 shows the circuit parameters of our test circuits. The first four circuits (*struct*, *biomed*, *avqsmall*, *avqlarge*) are the largest ones from the 1993 MCNC layout benchmark sets [24]. The results for *struct* and *biomed* were obtained *without* use of FMM. The next five circuits are from ISPD98 benchmark suite [25]. We tested our algorithms on all 18 circuits in the ISPD98 suite, but, due to page limitations, we cannot include all results here. We show results for five of these circuits that demonstrate our algorithm’s scalability. Both the solution-quality and runtime trends are the same on the circuits for which results are not shown.

We ran all experiments on a Sun UltraSparc-2/168MHz workstation with 512MB memory. After the global placement, we use DOMINO as the final placer. CPU times are in seconds; all times listed include the time spent in the final placement. Wire length of each net is computed as the half perimeter of the enclosing bounding box.

### 8.1 Impact of Nonlinear Programming

To measure the impact of constrained nonlinear programming on the final placement result, for circuit *biomed*, we vary the level for which we run the NP-engine, and get results as shown in Table 2, where level 0 means no NP-engine is run at all, while 1 means the NP-engine is run at the coarsest level, 2 means the NP-engine is run at the coarsest and the second coarsest levels, and so on.

In general, as the number of levels to which the NP-engine is applied increases, better final placement results are obtained at a cost of increased CPU time. Similar trends have been observed on other circuits.

Table 3: Comparison of Different Clustering Algorithms

circuit	MESC	MESCg	RandomMatch
struct	7.68e+05	7.64e+05	8.24e+05
biomed	3.86e+06	3.90e+06	4.09e+06
avqsmall	1.25e+07	1.29e+07	1.32e+07
avqlarge	1.37e+07	1.39e+07	1.44e+07

Table 4: Comparison with GORDIAN-L+DOMINO

circuit	MPL+Domino		GORDIAN-L vs. MPL	
	wirelength	CPU	wirelength	CPU
struct	7.68e+05	53.0	0.99	0.46
biomed	3.86e+06	336.9	1.01	0.69
avqsmall	1.25e+07	893.5	0.94	2.12
avqlarge	1.37e+07	814.0	0.95	2.56
ibm04	7.40e+06	1453.3	0.93	2.45
ibm07	1.10e+07	602.0	0.99	13.9
ibm09	1.27e+07	680.4	0.93	19.7
ibm14	4.33e+07	2965.7	0.94	20.5
ibm17	7.43e+07	8941.1	0.91	14.8

Note that the wirelength/runtime tradeoff curve is not completely monotone. For example, in Table 2, using two levels is slightly faster than using one level. The reason is that in this particular instance, the nonlinear programming problem happens to be easy: a few iterations take us to convergence. In general, we do not expect such CPU-time reduction.

### 8.2 Comparison of Clustering Algorithms

To measure the impact of different clustering algorithms, we ran our algorithm (MPL) under different clustering algorithms: MESC, MESCg, and RandomMatch. The results are shown in Table 3. Because all clustering algorithms have the same order, CPU time is not shown in Table 3. One can see that MESC and MESCg outperform RandomMatch on all cases by a large percentage. But, surprisingly, MESCg generally gives worse results than MESC, although the gap is very small. We believe the reason is that during the clustering procedure, an initial unconstrained placement gives us almost exclusively connectivity information. In view of this, we use MESC as our clustering algorithm during all other experiments.

### 8.3 Comparison with GORDIAN-L

We compared our algorithm with GORDIAN-L followed by DOMINO. We know that quadratic placement requires fixed pads. To ensure a fair comparison, we first ran GORDIAN-L+DOMINO on the test circuits and extracted the pads’ locations from the output. Then we ran MPL followed by DOMINO. The results are summarized in Table 4. Columns “GORDIAN-L vs. MPL” shows the results of GORDIAN-L+DOMINO divided by those obtained by our approach. Thus, values larger than 1.0 mean that our approach is better or faster.

Our approach gives results on average 4.9% worse than GORDIAN-L+DOMINO but uses significantly less CPU time, especially as the number of cells increases. For example, for the largest circuit *ibm17* with around 184K cells, our approach can obtain a placement in about 2.5 hours with relatively good quality, only 9.4% percent from the result of GORDIAN-L+DOMINO. This suggests our approach has better scalability and can be

used to achieve accurate placement estimation during earlier design phases.

## 9 Conclusions

By successfully integrating a nonlinear programming formulation into a multilevel framework, we have developed a very fast and scalable placement method that exploits several recent advances in scientific computation. This method holds great promise as a means of facilitating the timing convergence of synthesis and physical design. Many variations on our framework and its components are possible. In future work, we will investigate ways of improving solution quality, possibly at some cost in runtime, and we will work with industry to incorporate other design constraints (e.g., timing and noise) directly into our formulation. We will also gather data for comparison against more recent placement packages such as ITOOLS [26] whose results are significantly better than GORDIAN-L's.

## References

- [1] G. Sigl, K. Doll, and F. M. Johannes, "Analytical placement: A linear or a quadratic objective function?," in *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 427–432, 1991.
- [2] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer Academic Publishers, 1988.
- [3] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. on Computer-Aided Design*, pp. 349–359, March 1995.
- [4] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "Gordian: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 356–365, 1991.
- [5] Y. Sankar and J. Rose, "Trading quality for compile time: Ultra-fast placement for fpgas," in *FPGA '99, ACM Symp. on FPGAs*, pp. 157–166, 1999.
- [6] M. Sarrafzadeh and M. Wang, "Nrg: global and detailed placement," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 532–537, IEEE, 1997.
- [7] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. 35th ACM/IEEE Design Automation Conference*, pp. 269–274, 1998.
- [8] R. Tsay, E. S. Kuh, and C. Hsu, "Proud: A sea-of-gates placement algorithm," *IEEE Design and Test of Computers*, vol. 12, pp. 44–56, 1988.
- [9] W. L. Briggs, *A Multigrid Tutorial*. Philadelphia: SIAM, 1987.
- [10] A. Brandt, "Multi-level adaptive solutions to boundary value problems," *Mathematics of Computation*, vol. 31(138), pp. 333–390, 1977.
- [11] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proc. 34th ACM/IEEE Design Automation Conference*, pp. 526–529, 1997.
- [12] "Interior point methods online: <http://www-unix.mcs.anl.gov/otc/interiorpoint/archive.html>."
- [13] P. Chin and A. Vannelli, "Interior point methods for placement," *IEEE Int. Symp. on Circuits and Systems*, pp. 169–172, 1994.
- [14] T. Gao, P. M. Vaidya, and C. L. Liu, "A performance driven macro-cell placement algorithm," in *Proc. 29th ACM/IEEE Design Automation Conference*, pp. 147–152, 1992.
- [15] N. Quinn and M. Breuer, "A force-directed component placement procedure for printed circuit boards," *IEEE Trans. on Circuits and Systems CAS*, vol. CAS-26, pp. 377–388, 1979.
- [16] W. Murray and M. H. Wright, "Line search procedures for the logarithmic barrier function," *SIAM J. on Optimization*, vol. 4, number 2, pp. 229–246, 1994.
- [17] S. G. Nash, "Newton-type minimization via the Lanczos method," *SIAM J. on Numerical Analysis*, vol. 21, pp. 770–788, 1984.
- [18] Y. Saad, *Iterative methods for sparse linear systems*. Pacific Grove, California: PWS publishing, 1996.
- [19] M. Gu and S. Eisenstat, "A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem," *SIAM J. on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 172–191, 1995.
- [20] K. Nabors and J. White, "Fastcap: A multipole accelerated 3-d capacitance extraction program," *IEEE Trans. on Computer-Aided Design*, vol. 10, no. 11, pp. 1447–1459, 1991.
- [21] S. Goto, "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout," *IEEE Trans. on Circuits and Systems*, vol. 28, pp. 12–18, January 1981.
- [22] J. Cong and S. K. Lim, "Edge separability based circuit clustering with application to circuit partitioning," in *Asia South Pacific Design Automation Conference, Yokohama Japan*, pp. 429–434, 2000.
- [23] H. Nagamochi and T. Ibaraki, "Computing edge connectivity in multigraphs and capacitated graphs," *SIAM Journal on Discrete Math.*, pp. 54–66, 1992.
- [24] K. Kozminski, "Benchmarks for layout synthesis," in *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 265–270, 1991.
- [25] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proc. Intl Symposium on Physical Design*, pp. 80–85, 1998.
- [26] "<http://www.twolf.com>."