Multilevel Generalized Force-directed Method for Circuit Placement *

Tony Chan+, Jason Cong†, Kenton Sze+
† UCLA Computer Science Department cong@cs.ucla.edu
+ UCLA Mathematics Department {chan,nksze}@math.ucla.edu

ABSTRACT

Automatic circuit placement has received renewed interest recently given the rapid increase of circuit complexity, increase of interconnect delay, and potential sub-optimality of existing placement algorithms [13]. In this paper we present a generalized force-directed algorithm embedded in mPL2's [12] multilevel framework. Our new algorithm, named mPL5, produces the shortest wirelength among all published placers with very competitive runtime on the IBM circuits used in [29]. The new contributions and enhancements are: (1) We develop a new analytical placement algorithm using a density constrained minimization formulation which can be viewed as a generalization of the force-directed method in [16]; (2) We analyze and identify the advantages of our new algorithm over the force-directed method; (3) We successfully incorporate the generalized force-directed algorithm into a multilevel framework which significantly improves wirelength and speed. Compared to Capo9.0, our algorithm mPL5 produces 8% shorter wirelength and is 2X faster. Compared to Dragon3.01, mPL5 has 3% shorter wirelength and is 12X faster. Compared to Fengshui5.0, it has 5% shorter wirelength and is 2X faster. Compared to the ultra-fast placement algorithm: FastPlace, mPL5 produces 8% shorter wirelength but is 6X slower. A fast mode of mPL5 (mPL5-fast) can produce 1% shorter wirelength than FastPlace1.0 and is only 2X slower. Moreover, mPL5-fast has demonstrated better scalability than FastPlace1.0.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*placement and routing*; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

*Financial supports from Semiconductor Research Consortium Contract 2003-TJ-1019, National Science Foundation grants ACI-0072112 and CCF-0430077, and Office of Naval Research grant N00014-03-1-0888 are gratefully acknowledged.

ISPD'05, April 3–6, 2005, San Francisco, California, USA.

General Terms

Algorithms, Design

Keywords

Standard Cell Placement, Multilevel, Force-directed Method

1. INTRODUCTION

Automatic circuit placement has received renewed interest recently given the rapid increase of circuit complexity, increase of interconnect delay, and potential sub-optimality of existing placement algorithms [13]. As integrated circuit technology further scales, the design sizes are getting larger. Recently, [15] shows the importance of building a good physical hierarchy from a flattened or nearly flattened logical netlist for performance optimization. Therefore, large-scale placement on a nearly flattened netlist is needed for physical hierarchy generation to achieve the best performance. This is even more critical for deep sub-micron or nanometer designs as the interconnect has become the performance bottleneck.

Typical techniques used in the current state-of-the-art placement tools consist of min-cut partitioning [7, 31], simulated annealing [30, 14], analytical methods with quadratic wirelength minimization [21, 11], linear wirelength minimization [22, 12] and log-sum-exponential wirelength minimization [26, 18]. The first two techniques always produce a global placement with not much cell overlapping. On the other hand, analytical techniques for minimizing some unconstrained smoothed wirelength objectives usually introduce a lot of cell overlapping or area congestion during the global placement. Hence, area congestion removal techniques such as slot assignment [11, 12], recursive bisection/quadrisection partition [32], ripple-move [17, 12], cell shifting [29] and grid warping [33] have been introduced. Algorithms [18, 16] are also proposed to minimize the wirelength objective and area congestion simultaneously. Usually, those placement techniques are embedded in a hierarchical/multilevel framework to speed up the placement process [18, 7, 30, 31, 11, 12, 14].

In this paper, we use the multilevel technique [11, 12, 14, 9, 18] combining with a new analytical method for large-scale placement. Multilevel/multigrid methods have been successfully applied to solve partial differential equations [5, 6]. Also, they have been successfully used to solve the hypergraph partitioning problem [19, 2]. We use the multilevel algorithm because it gives better scalability and better global optimization.

Our novel analytical placement algorithm is based on a mathematically sound foundation for supporting the density constraint, and can be viewed as a generalization of the force-directed method in [16]. In [16], it uses a quadratic wirelength objective and adds forces on cells based on area density of the placement. Adding forces on cells is equivalent to modifying the right-hand side of the linear system arising from the quadratic wirelength minimization. Hence each iteration can be solved easily. However, it suffers from the inaccurate approximation by quadratic wirelength objective as illustrated in [20, 22, 24] and the ad hoc scaling of the spreading forces for supporting the density constraint.

The new contributions and enhancements presented in this paper are as follows:

- We develop a new analytical placement algorithm using a density constrained minimization formulation which can be viewed as a generalization of the force-directed method in [16].
- We analyze and identify the advantages of our new algorithm over the force-directed method in [16].
- We successfully incorporate the generalized force-directed algorithm into a multilevel framework which significantly improves the wirelength and speed. We use the multilevel framework proposed in mPL2 [12]. The new algorithm is named mPL5.

The remainder of this paper is organized as follows. Section 2 presents the constrained minimization formulation and how it is related to the force-directed method [16]. A generalized force-directed algorithm is proposed. Section 3 describes the multilevel framework and how the generalized force-directed algorithm is incorporated into the multilevel paradigm. In Section 4, experimental results are presented to demonstrate the effectiveness of our new algorithm. Comparisons with other state-of-the-art placers are given. Finally, the conclusion and future work are given in Section 5.

2. MATHEMATICAL MODEL

In this section we give an introduction to the circuit placement problem and present a nonlinear constrained optimization problem formulation for it.

2.1 Circuit Placement Problem Formulation

The circuit placement problem can be characterized as a hypergraph placement problem. Let $H = (V, E)$ be the hypergraph. Let $V = \{v_1, v_2, \ldots, v_N, v_{N+1}, \ldots, v_{N+P}\}$ represent the set of cells/modules and $E = \{e_1, e_2, \ldots, e_m\}$ represent the set of nets/interconnects. The set $\{v_{N+1}, \ldots, v_{N+P}\}$ represents pads (fixed terminals) which are fixed throughout placement, and each e_i is a subset of V that gives the connection/relation among the cells. Net with degree k is called $k-$pin net.

Let (x_k, y_k) be the center coordinate of the cell v_k. The wirelength of a net e, given by

$$l(e) = \max_{v_i, v_j \in e, i<j} |x_i - x_j| + \max_{v_i, v_j \in e, i<j} |y_i - y_j|, \quad (1)$$

is the half-perimeter of the smallest rectangle containing all the cells in e. The total wirelength of a given circuit is the sum of the wirelength of each net in E. Our objective is to place the cells, subject to some constraint such as cell non-overlapping, such that the total wirelength $\sum_{e \in E} l(e)$ is minimized. Currently, we assume standard cell placement to be that of all the cells having the same height and a number of standard rows is given. The cell non-overlapping constraint for the standard cell placement is to place all the movable cells on the given rows without overlapping each other. However, our algorithm can be easily generalized to the case where we have cells of different heights.

Typically, the placement problem is divided into two stages: global placement and detailed placement. For global placement, one needs to place the cells evenly distributed on the placement region where cell overlapping is allowed. The global placement solution is then legalized by discrete methods in detailed placement, which can also further reduce the wirelength by local cell swapping without creating cell overlapping. In this paper, we mainly focus on global placement.

2.2 Constrained Minimization Formulation

In this section we present a constrained minimization formulation for the placement problem. We discuss smooth approximations to the wirelength objective eq(1) and smooth approximation to the pairwise non-overlapping constraint for the standard cell placement problem.

2.2.1 Smooth Wirelength Approximation

Since eq(1) is not differentiable and the constraints are highly non-convex, the minimizer is hard to locate. Therefore, using continuous differentiable functions to approximate eq(1) is necessary. Many studies, for example [21, 16, 11], use a quadratic function approximation given by

$$\sum_{e \in E} \Big(\sum_{v_i, v_j \in e, i<j} w_{ij} |x_i - x_j|^2 + \sum_{v_i, v_j \in e, i<j} w_{ij} |y_i - y_j|^2 \Big). \quad (2)$$

The advantage of using the quadratic wirelength objective is that its unconstrained minimizer can be obtained by solving a positive definite linear system of equations. However, it over-penalizes the long nets which gives a bad half-perimeter wirelength placement solution.

In this paper we use the following better half-perimeter wirelength approximation objective

$$\begin{aligned} &\eta \sum_{e \in E} \Big(\log \sum_{v_k \in e} \exp(x_k/\eta) + \log \sum_{v_k \in e} \exp(-x_k/\eta) \\ &+ \log \sum_{v_k \in e} \exp(y_k/\eta) + \log \sum_{v_k \in e} \exp(-y_k/\eta) \Big) \end{aligned} \quad (3)$$

proposed in [26] and recently used in Aplace [18], where the smaller η, the more accurate the approximation. However, we can not choose too small η due to machine precision and numerical stability. In experiments, we scale the placement problem so that all the cell locations are between 0 and 1. η is then set to 0.01.

We have also proposed and studied another approximation to eq(1) using L_p-norm:

$$\sum_{e \in E} \Big(\big(\sum_{v_k \in e} x_k^p \big)^{\frac{1}{p}} - \big(\sum_{v_k \in e} x_k^{-p} \big)^{-\frac{1}{p}} + \big(\sum_{v_k \in e} y_k^p \big)^{\frac{1}{p}} - \big(\sum_{v_k \in e} y_k^{-p} \big)^{-\frac{1}{p}} \Big) \quad (4)$$

since the first term and the second term tend to $\max\{x_k\}$ and $\min\{x_k\}$ respectively as p tends to infinity. We set $p = 32$ in experiments so that x^p and $x^{\frac{1}{p}}$ can be computed efficiently. Numerical results verifying the effectiveness of

different objectives are given in Table 4. We remark that a slightly different approximation using L_p-norm is proposed in [20].

2.2.2 Smooth Constraints Approximation

Since the pairwise non-overlapping constraints are highly nonconvex and difficult to satisfy during the global placement, we replace the constraints by bin density constraints discussed in the following.

Based on the placement region R, we divide the region into $m \times n$ uniform non-overlapping sub-regions (bins) $B_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$ such that $\cup_{i,j} B_{ij} = R$. Let h_x and h_y be the bin width and bin height respectively. Define D_{ij} to be the average density in the bin B_{ij} which is given by

$$D_{ij}(\mathbf{x}, \mathbf{y}) = \sum_{k=1} a_{ij}(v_k)/(h_x h_y), \tag{5}$$

where $a_{ij}(v_k)$ is the fractional area of cell v_k lying inside bin B_{ij} (see Figure 1).

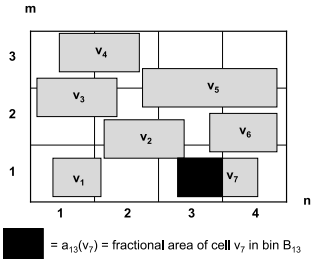

Figure 1: Illustration of fractional cell area in a 3x4 bins region.

We consider the constrained minimization problem:

$$\begin{array}{ll} \min & W(\mathbf{x}, \mathbf{y}) \\ s.t. & D_{ij} = K, \quad 1 \leq i \leq m, 1 \leq j \leq n \end{array} \tag{6}$$

where D_{ij} is the average density in B_{ij} defined through eq(5) and K is the total cells area divided by the area of the placement region R[1]. The current problem is to find a placement that minimizes the wirelength $W(\mathbf{x}, \mathbf{y})$ such that the cells are evenly distributed over the region. However, it is difficult to solve the above problem since the density function is not differentiable. To make the problem easier to solve, we use the inverse Laplace transformation [10] to smooth the density function. The smoothing operator $\Delta_\epsilon^{-1} d(x, y)$ is defined by solving the following Helmholtz equation:

$$\begin{cases} \Delta\psi(x,y) - \epsilon\psi(x,y) = d(x,y), & (x,y) \in R \\ \frac{\partial\psi}{\partial\nu} = 0, & (x,y) \in \partial R \end{cases} \tag{7}$$

where $\epsilon > 0$, ν is the outer unit normal, ∂R is the boundary of R, $d(x, y)$ is the continuous density function and Δ is a differential operator given by

$$\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{8}$$

The inverse operator is well defined, as eq(7) has a unique solution for any $\epsilon > 0$. Since the solution of eq(7) gains two more derivatives [10] than $d(x, y)$, ψ is a smoothed version of the density function.

We use the finite difference method [25] to discretize the problem eq(7) using the bin grids we defined above. The Neumann boundary condition is used for the discretization scheme. Let $\psi_{i,j}$ be the value of ψ at the center of the bin B_{ij}. The approximation scheme is given by

$$\begin{array}{l} \frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h_y^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{h_x^2} \\ -\epsilon\psi_{i,j} = D_{ij}, \quad \forall\ 1 \leq i \leq m, 1 \leq j \leq n \end{array} \tag{9}$$

where

$$\begin{array}{l} \psi_{0,j} = \psi_{1,j} \quad \forall\ 1 \leq j \leq n \\ \psi_{m+1,j} = \psi_{m,j} \quad \forall\ 1 \leq j \leq n \\ \psi_{i,0} = \psi_{i,1} \quad \forall\ 1 \leq i \leq m \\ \psi_{i,n+1} = \psi_{i,n} \quad \forall\ 1 \leq i \leq m, \end{array} \tag{10}$$

and D_{ij} is the average density in B_{ij}. Let L_ϵ be the matrix corresponding to the above linear system. Then $\Psi = (\psi_{11}, \psi_{12}, \ldots, \psi_{mn})^t$ can be computed by solving the following linear system

$$L_\epsilon \Psi = \mathbf{D}, \tag{11}$$

where $\mathbf{D} = (D_{11}, D_{12}, \ldots, D_{mn})^t$. Note that the problem eq(11) can be solved in $O(mn \log mn)$ by fast discrete cosine transform [8, 27].

Now we can reformulate the problem eq(6) as

$$\begin{array}{ll} \min & W(\mathbf{x}, \mathbf{y}) \\ s.t. & \psi_{ij} = \bar{K}_\epsilon, \quad 1 \leq i \leq m, 1 \leq j \leq n \end{array} \tag{12}$$

where $\Psi = L_\epsilon^{-1}\mathbf{D}$ and $\bar{K}_\epsilon \mathbf{1} = L_\epsilon^{-1} K \mathbf{1} = -K/\epsilon \mathbf{1}$ is a constant vector where $\mathbf{1} = (1, \ldots, 1)^t$. In next section, we discuss how to solve the above problem and how it is related to the force-directed method [16].

2.3 Problem Solver

There are many nonlinear programming techniques to solve eq(12). We use the Uzawa algorithm [4] to solve eq(12). The advantage is that it does not require a Hessian inversion to find a minimizer satisfying the KKT condition. Another reason is that the iterative scheme can be viewed as a generalization of the force-directed method [16]. By applying the Uzawa algorithm to solve eq(12) through the Lagrange multiplier, we get the following iterative scheme:

$$\begin{cases} \nabla W(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}) + \sum_{i,j} \lambda_{ij}^k \nabla \psi_{ij} = 0 \\ \lambda_{ij}^{k+1} = \lambda_{ij}^k + \alpha(\psi_{ij} - \bar{K}_\epsilon) \end{cases} \tag{13}$$

where λ^k is the Lagrange multiplier at $k-$th iteration, α is a parameter to control the rate of convergence, and x^k and y^k are the cell locations at the $k-$th iteration.

The gradient of ψ_{ij} with respect to cell v_k can be approximated by the difference scheme

$$\nabla_{x_k} \psi_{ij} = \frac{\psi_{i,j+1} - \psi_{i,j}}{h_x} \quad \text{and} \quad \nabla_{y_k} \psi_{ij} = \frac{\psi_{i+1,j} - \psi_{i,j}}{h_y} \tag{14}$$

if the center of cell v_k is inside B_{ij} and zero otherwise.

In [16], it derives that the divergence of the forces $f(x, y)$ is proportional to the density; that is,

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} = c \cdot d(x, y), \tag{15}$$

[1]Note that in general we may set different density target K_{ij} for bin B_{ij} to reflect uneven density requirements due to pre-placed blocks etc. For all benchmarks we tested in the work, K is a constant for all bins.

where c is a constant. Also, there exists a scalar function $\phi(x, y)$ satisfying

$$\nabla\phi(x, y) = f(x, y). \tag{16}$$

Combining eq(15) and eq(16) gives the following equation

$$\Delta\phi(x, y) = c \cdot d(x, y) \tag{17}$$

with boundary conditions that the magnitude of the forces $\nabla\phi(x, y)$ is zero at infinity.

Comparing eq(7) with eq(17), the main difference is the boundary condition if we choose small ϵ. The boundary condition in our formulation eq(7) tells that the forces pointing outside the boundary are zero, which makes more sense than assuming the forces being zero at infinity as in [16] since we want to place the cells inside a finite region.

Moreover, the force-directed method in [16] can be considered a special case of eq(13). It uses the quadratic wirelength objective eq(2) for $W(\mathbf{x}, \mathbf{y})$ and iteratively solves

$$\begin{pmatrix} C & 0 \\ 0 & C \end{pmatrix} \begin{pmatrix} \mathbf{x^{k+1}} \\ \mathbf{y^{k+1}} \end{pmatrix} + \begin{pmatrix} \mathbf{p_x} \\ \mathbf{p_y} \end{pmatrix} + \tau_k \begin{pmatrix} \mathbf{f_x^k} \\ \mathbf{f_y^k} \end{pmatrix} = 0 \tag{18}$$

until all cells are well distributed over the chip region. C, $\mathbf{p_x}$ and $\mathbf{p_y}$ are derived from $\nabla W(\mathbf{x}, \mathbf{y})$. The τ_k is a scalar to control the movement of cells in each iteration. The horizontal force $\mathbf{f_x^k}$ and the vertical force $\mathbf{f_y^k}$ acting on the cells are given by $\sum(\nabla_{x_1}\phi_{ij}, \ldots, \nabla_{x_N}\phi_{ij})^t$ and $\sum(\nabla_{y_1}\phi_{ij}, \ldots, \nabla_{y_N}\phi_{ij})^t$ respectively computed based on the placement solution at the $k-$th iteration. Clearly, this is a particular case of eq(13) by setting $\lambda_{ij}^k = \tau^k$. One can expect the above fixed point iteration requiring a small enough τ_k for convergence. But we know λ^k is the Lagrange multiplier for eq(12) which has to be large enough to get a well-distributed placement. Also, the λ^k is a vector in eq(13) and has each of its component acting as a scaling factor for the forces induced in the corresponding bins. These show that our new algorithm is more general and robust, and, overcomes the shortcoming of ad hoc force scaling selection used in [16].

In each step of the iterative scheme eq(13), we have to solve a nonlinear equation which can be solved by the time marching scheme [28, 3]. The solution of the nonlinear equation is a steady solution of the following ordinary differential equation (ODE):

$$\begin{cases} \begin{pmatrix} \frac{\partial \mathbf{x(t)}}{\partial t} \\ \frac{\partial \mathbf{y(t)}}{\partial t} \end{pmatrix} = -(\nabla W(\mathbf{x(t)}, \mathbf{y(t)}) + \sum_{\mathbf{i,j}} \lambda_{\mathbf{ij}} \nabla \psi_{\mathbf{ij}}) \\ (\mathbf{x(0)}, \mathbf{y(0)}) \text{ is a given initial placement,} \end{cases} \tag{19}$$

where $(\mathbf{x(t)}, \mathbf{y(t)})$ denotes the placement at time t. It can be considered a gradient descent scheme for the Lagrangian function

$$L(\mathbf{x}, \mathbf{y}, \lambda) = \mathbf{W}(\mathbf{x}, \mathbf{y}) + \sum_{\mathbf{i,j}} \lambda_{\mathbf{ij}}(\psi_{\mathbf{ij}} - \bar{\mathbf{K}}_\epsilon) \tag{20}$$

since

$$\frac{\mathrm{d}L(\mathbf{x}, \mathbf{y}, \lambda)}{\mathrm{dt}} = -\|\frac{\partial(\mathbf{x}, \mathbf{y})}{\partial \mathrm{t}}\|^2 < 0. \tag{21}$$

One can think of it as minimizing the wirelength objective and constraints penalty simultaneously at each iteration. We solve the above ODE by the explicit Euler method [25].

The algorithm we used to solve eq(12) is given in Table 1. It is called GFD (Generalized Force-directed) algorithm. The algorithm takes in the number of outer iterations and the stopping criterion for inner iteration. α is a parameter to speed up the convergence. γ is the increasing rate for α. β is the percentage of non-zero density bins. N is the number of movable cells, and P is the number of pads. Since one can only get a local minimizer by solving eq(13), the initial solution is important. The outer iterations can be considered a continuation method where the solution at each outer iteration is used as an initial solution for the next iteration.

We use uniform bin grids, and the number of bins is roughly equal to the number of cells.

We remark that Aplace [18] uses a penalty method to solve eq(6). It uses a bell shape function [26] to smooth the density constraint *locally*. In our case, however, the inverse Laplace transformation eq(7) smoothes the density function *globally*.

Since the global placement produced by the GFD algorithm may contain cell overlapping, a discrete algorithm is used to legalize the solution. We use a simple greedy algorithm [23] to place the cells in standard rows without overlapping. Local greedy cell swapping, where each move does not create overlapping, is then applied to reduce wirelength.

GFD(*outer_iters*, *stop_percent*)
if initial placement not given
  use the unconstrained minimizer of the quadratic
  wirelength objective as an initial solution.
endif
compute nnb = number of non-zero density bins.
set P = total number of pads.
set N = total number of cells.
set $inner\_iters = N$.
set $\gamma = 1.5$. (Experiments show that it is a good
trade-off between runtime and wirelength)
for $i = 1$ to $outer\_iters$
  set $\alpha = \frac{\sqrt{P}}{h_x h_y \log N}$.
  $\beta = \min\{\frac{100i}{outer\_iters}, stop\_percent\}$.
  $\lambda = 0$.
  for $j = 1$ to $inner\_iters$
    if nnb not increased
      $\alpha = \gamma\alpha$.
    endif
    $\lambda = \lambda - \alpha(\psi - K_\epsilon)$.
    solve the ODE eq(19) by explicit Euler method.
    compute nnb.
    if more than β% non-zero density bins
      break.
    endif
  endfor
call detailed placement.
endfor

Table 1: GFD algorithm.

3. MULTILEVEL FRAMEWORK

Many studies [11, 12, 14, 9] show that multilevel algorithm is a promising technique to handle large-scale problems. It is not only used for speed-up, but also for better global optimization.

In this section we incorporate the GFD algorithm into the multilevel framework that is used in mPL [12]. The

multilevel framework consists of coarsening, interpolation, relaxation, and multilevel flow. We review and discuss each component in the following sections.

3.1 Coarsening

The purpose of coarsening is to build a hierarchy for the multilevel paradigm. We use a modified first-choice (FC) hypergraph coarsening based on the FC first proposed in [19]. We define the affinity between vertex v and w as

$$r_{vw} = \sum_{e \in E | v,w \in e} \frac{w(e)}{(|e|-1)\text{area}(e)}, \quad (22)$$

where $w(e)$ is the weight assigned to net e, area(e) denotes the sum of the areas of the cells in e, and $|e|$ denotes the number of cells in net e.

The vertices are first ordered in descending order of the vertex degree. Vertices are then examined sequentially, and the affinities eq(22) each vertex v has for vertices with which it shares hyperedges are computed. An affinity graph is then constructed by joining each vertex to exactly one of its neighbors for which it has maximal affinity. Each group of joined vertices is called a cluster and become the coarser level vertex. Hyperedges are defined on the clusters in the obvious way: each hyperedge on the finer level becomes a hyperedge (the set of clusters containing those vertices) at the coarser level, with the singleton hyperedge simply ignored. We hence get a smaller hypergraph at the coarser level.

3.2 Relaxation

mPL3 [12] solves nonlinear programming at the coarsest level. On the subsequent levels, it uses the GOTO swapping and quadratic relaxation on subsets with ripple-move to relieve area congestion. We replace those optimization techniques with the GFD algorithm. It is a more powerful relaxation, as it moves all cells simultaneously to reduce wirelength subject to the area density constraint.

3.3 Interpolation

Interpolation is used to transfer solutions from level to level. For example, given a placement solution at the coarse level, we use it to compute the placement solution at the finer level via interpolation.

A graph model of connectivity is employed to define the interpolation: the weight of edge e_{ij} is

$$w(e_{ij}) = \sum_{\{e \in E \mid i,j \in e\}} \frac{w(e)}{(|e|-1)}. \quad (23)$$

For efficiency, only weights above a certain threshold (currently 1/4) are used. Finer-level vertices v_i within each cluster with the highest vertex degree (using cell area to break the tie) are designated as "C-points" and are given the positions of their parent clusters. "C-point" locations are fixed during interpolation. The remaining points are designated as "F-points" and are placed at the weighted average of the positions of the C-points to which they are connected. Once an F-point has been placed, it can be treated like a C-point and used to influence the positioning of other F-points to which it has connections. Moreover, since the process depends on the vertex order, iterations are used to allow all interconnected nodes to influence each others' positions. For this purpose, the nodes are ordered by decreasing connectivity $w(v_i) = \sum_j w(e_{ij})$, following eq(23).

3.4 Multilevel Flow

After the first V-cycle, an additional V-cycle is used to improve the result. During the reaggregation phase, spatial proximity is used in the FC affinity along with netlist connectivity. We re-define the affinity between vertex v and w to be

$$r_{vw} = \sum_{e \in E | v,w \in e} \frac{w(e)}{(|e|-1)\text{area}(e)\text{dist}(v,w)}, \quad (24)$$

where dist(v,w) is the Euclidean distance between v and w. Thus, clusters are placed at the weighted average of their components' positions, the weights identical to those used in the interpolation eq(23). Relaxation on this modified hierarchy is then used to further reduce the wirelength.

use modified FC (c.f. eq(22)) to coarsen the hypergraph until the number of cells < 500.
set nl = number of levels.
set $stop\_percent = 97$
% suppose level nl is the finest level corresponding
% to the original hypergraph.
for $i = 1$ to $nl - 1$
  set $distri\_percent = min(50 + 50 * i/nl, 90)$.
  at level i,
    call GFD$(1, distri_percent)$.
  interpolate placement from level i to level $i + 1$.
endfor
% start the second V-cycle.
use modified geometric based FC (c.f. eq(24))
to coarsen the hypergraph until the number of
cells < 500.
placement from first V-cycle is interpolated to coarse
levels during the coarsening.
set nl = number of levels.
for $i = 1$ to $nl - 1$
  set $distri\_percent = min(50 + 50 * i/nl, 90)$.
  at level i,
    call GFD$(1, distri_percent)$.
  interpolate placement from level i to level $i + 1$.
endfor
call GFD$(1, stop_percent)$.
call detailed placement.

Table 2: mPL5 algorithm.

The multilevel GFD algorithm (mPL5) is shown in Table 2. Figure 2 shows the multilevel flow in mPL5. A fast mode of mPL5 is obtained by: (1) set the $stop\_percent = 95$; (2) increase α in the GFD algorithm whether nnb is increased or not; (3) reduce the number of bins to half of the normal; (4) reduce cell swapping in the detailed placement.

4. NUMERICAL RESULTS

The benchmarks used in our experiments are the same as in [29], and are provided by the authors of FastPlace1.0 [29]. They are originally derived from the ISPD-02 suite downloaded from [1]. The macro blocks are modified to be standard cells in a way that the height of macro blocks is

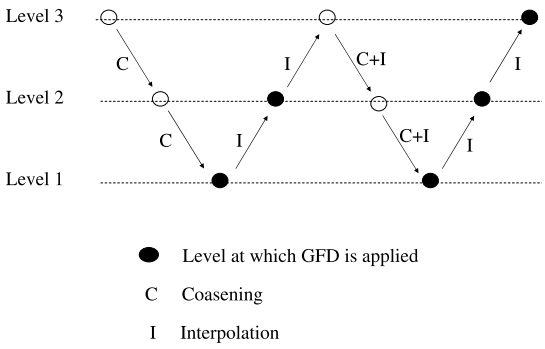

Figure 2: Multilevel flow of mPL5.

brought down to the standard cell height and the width of macro blocks, if exceeding 4X average width, is changed to a value of 4X average width.

Circuit	GFD(20) WL,runtime	GFD(30) WL,runtime	mPL5 WL,runtime
ibm01	0.96 , 1.34	0.93 , 1.75	0.87 , 0.58
ibm02	0.84 , 1.31	0.82 , 1.66	0.78 , 0.67
ibm03	0.97 , 1.39	0.95 , 1.84	0.94 , 0.56
ibm04	0.93 , 1.38	0.90 , 1.74	0.76 , 0.63
ibm05	0.93 , 1.13	0.83 , 1.14	0.63 , 0.57
ibm06	0.96 , 1.47	0.90 , 1.70	0.76 , 0.58
ibm07	0.95 , 1.52	0.92 , 2.20	0.77 , 0.43
ibm08	0.96 , 1.37	0.93 , 1.78	0.86 , 0.45
ibm09	0.94 , 1.48	0.93 , 1.91	0.83 , 0.44
ibm10	0.89 , 1.35	0.86 , 1.73	0.77 , 0.41
ibm11	0.92 , 1.23	0.89 , 1.73	0.77 , 0.39
ibm12	0.95 , 1.41	0.90 , 1.75	0.87 , 0.47
ibm13	0.95 , 1.45	0.92 , 1.92	0.80 , 0.41
ibm14	0.94 , 1.47	0.93 , 1.94	0.77 , 0.34
ibm15	0.96 , 1.47	0.94 , 1.94	0.80 , 0.33
ibm16	0.96 , 1.39	0.93 , 1.73	0.78 , 0.34
ibm17	0.95 , 1.37	0.91 , 1.81	0.71 , 0.37
ibm18	0.93 , 1.33	0.89 , 1.72	0.69 , 0.34
average	0.94 , 1.38	0.90 , 1.78	0.79 , 0.46

Table 3: Relative wirelength(WL) and runtime of GFD(20), GFD(30) and mPL5 with respect to GFD(10).

All the experiments are run on a Linux, 2.4GHz machine. In Table 3, we run the GFD algorithm with a different number of *outer_iters* shown in the parenthesis. It also shows comparisons with mPL5, the multilevel GFD. The wirelength and runtime are relative to GFD(10), the GFD algorithm with 10 outer iterations. The *stop_percent* in the GFD algorithm is set to 97 in the comparisons. We can see the wirelength is getting shorter as we increase the number of outer iterations. We remark that keeping an increase in the number of outer iterations, though increasing the runtime, does not further significantly reduce the wirelength. However, the multilevel GFD (mPL5) significantly outperforms the GFD both in quality and runtime. This shows that our multilevel algorithm is a very effective technique that gives better scalability and better global optimization. Figure 3 shows the placement solutions of mPL5 at each level in the second V-cycle. We can see that cells are distributed more evenly from level to level.

Circuit	LogSumExp WL,runtime(s)	Lp-norm WL,runtime	quadratic WL,runtime
ibm01	1.57E+06 , 45	1.05 , 1.71	1.73 , 0.81
ibm02	3.51E+06 , 66	1.02 , 1.80	1.84 , 1.65
ibm03	4.83E+06 , 66	0.99 , 1.82	1.63 , 0.64
ibm04	5.90E+06 , 117	0.98 , 1.47	1.48 , 0.47
ibm05	9.85E+06 , 94	1.06 , 1.74	1.49 , 1.17
ibm06	4.73E+06 , 161	1.03 , 1.36	1.82 , 0.50
ibm07	8.14E+06 , 209	1.04 , 1.62	1.50 , 0.67
ibm08	9.49E+06 , 321	1.01 , 1.45	1.79 , 0.72
ibm09	9.25E+06 , 313	1.05 , 1.53	1.65 , 0.53
ibm10	1.74E+07 , 302	1.03 , 2.07	1.47 , 0.72
ibm11	1.39E+07 , 379	1.04 , 1.62	1.54 , 0.52
ibm12	2.31E+07 , 367	1.02 , 1.68	1.34 , 0.67
ibm13	1.67E+07 , 404	1.03 , 1.72	1.69 , 0.63
ibm14	3.25E+07 , 1164	1.03 , 1.46	1.60 , 0.71
ibm15	3.92E+07 , 1250	1.04 , 1.84	1.61 , 0.80
ibm16	4.32E+07 , 1387	1.04 , 1.63	1.66 , 0.79
ibm17	6.27E+07 , 1347	1.03 , 1.73	1.42 , 0.85
ibm18	4.18E+07 , 1502	1.07 , 1.78	1.77 , 0.97
average	1.00 , 1.00	1.03 , 1.67	1.61 , 0.77

Table 4: Comparisons of different objectives using half perimeter wirelength (WL) and runtime of global placement. Wirelength and runtime of Lp-norm ($p = 32$) and quadratic are divided by those of LogSumExp respectively.

In Table 4, we compare the performances of different objectives: LogSumExp eq(3), Lp-norm eq(4) and quadratic eq(2) under the mPL5 platform. It shows that the global placement wirelength by Lp-norm is 3% longer than LogSumExp and with a 67% longer runtime. The runtime for quadratic is around 20% shorter than LogSumExp, but its wirelength is 61% longer. This demonstrates that the LogsumExp gives the best approximation to eq(1).

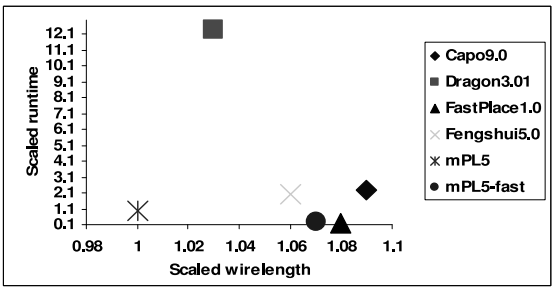

Figure 4: Wirelength and runtime comparisons on FastPlace1.0 IBM circuits.

In Table 5, we compare mPL5 with current state-of-the-art placers: Capo9.0 [7], Dragon3.01 [30], FastPlace1.0 [29] and Fengshui5.0 [31]. We are also interested in comparing mPL5 with Aplace [18]; however, Aplace's binary is not available for download. All the placers are run in default mode. Table 5 shows that overall mPL5 produces the shortest wirelength. Compared to Capo9.0, mPL5 has 8% shorter

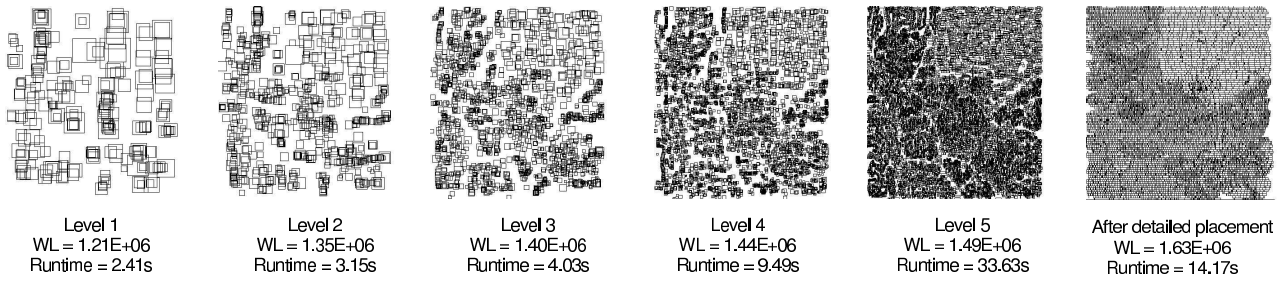

Figure 3: Placement solution at each level in the second V-cycle of mPL5.

Circuit	mPL5 Wirelength(WL)	mPL5 runtime(s)	Capo9.0 WL,runtime	Dragon3.01 WL,runtime	FastPlace1.0 WL,runtime	FengShui5.0 WL,runtime	mPL5-fast WL,runtime
ibm01	1.67E+06	64	1.08 , 1.90	1.02 , 16.81	1.09 , 0.10	1.08 , 2.06	1.09 , 0.23
ibm02	3.62E+06	126	1.09 , 1.82	1.02 , 7.34	1.06 , 0.13	1.02 , 1.93	1.02 , 0.26
ibm03	4.57E+06	113	1.10 , 2.56	1.05 , 8.04	1.12 , 0.13	1.03 , 2.39	1.05 , 0.28
ibm04	5.75E+06	151	1.06 , 2.47	1.00 , 10.95	1.04 , 0.13	1.05 2.16	1.04 , 0.35
ibm05	9.92E+06	158	1.02 , 2.52	0.98 , 17.36	1.05 , 0.14	1.00 , 2.29	1.05 , 0.30
ibm06	5.10E+06	200	1.11 , 2.01	0.98 , 10.44	1.04 , 0.12	1.02 , 2.12	1.03 , 0.28
ibm07	8.23E+06	259	1.11 , 2.45	1.04 , 9.61	1.08 , 0.19	1.09 , 2.32	1.09 , 0.30
ibm08	9.38E+06	389	1.05 , 1.73	0.96 , 15.47	1.02 , 0.14	n/a	1.05 , 0.38
ibm09	9.33E+06	342	1.08 , 2.30	1.07 , 16.26	1.12 , 0.17	1.06 , 1.72	1.07 , 0.28
ibm10	1.73E+07	450	1.10 , 2.42	1.04 , 10.96	1.07 , 0.20	1.07 , 1.56	1.11 , 0.30
ibm11	1.40E+07	437	1.09 , 2.70	1.03 , 7.81	1.09 , 0.19	1.04 , 2.31	1.06 , 0.30
ibm12	2.23E+07	482	1.11 , 2.48	1.03 , 11.15	1.08 , 0.20	1.07 2.03	1.06 , 0.36
ibm13	1.66E+07	596	1.10 , 2.32	1.05 , 7.73	1.11 , 0.20	1.09 , 1.80	1.07 , 0.28
ibm14	3.16E+07	1064	1.10 , 2.49	1.05 , 10.65	1.11 , 0.21	1.04 , 1.20	1.08 , 0.33
ibm15	3.85E+07	1379	1.09 , 2.41	1.04 , 11.14	1.13 , 0.23	1.07 , 2.15	1.07 , 0.30
ibm16	4.30E+07	1577	1.10 , 2.29	1.05 , 11.09	1.07 , 0.23	1.09 , 2.18	1.08 , 0.30
ibm17	6.13E+07	1705	1.09 , 2.32	1.08 , 22.22	1.08 , 0.23	1.08 , 2.17	1.09 , 0.30
ibm18	4.10E+07	1904	1.09 , 2.00	1.02 , 17.84	1.10 , 0.25	1.04 , 2.07	1.07 , 0.29
average	1.00	1.00	1.09 , 2.29	1.03 , 12.38	1.08 , 0.18	1.06 , 2.03	1.07 , 0.30

Table 5: Comparisons between mPL5, mPL5-fast, Capo9.0, Dragon3.01, FastPlace1.0 and Fengshui5.0. Program failure is denoted by n/a.

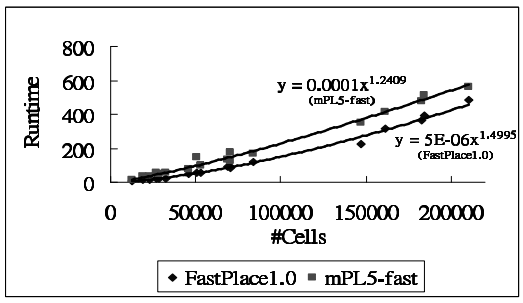

Figure 5: Scalability plot of FastPlace1.0 and mPL5-fast on IBM circuits.

wirelength and is 2X faster. Compared to Dragon3.01, mPL5 produces 3% shorter wirelength and is 12X faster. Compared to FastPlace1.0, it outperforms the wirelength by 8% but is 6X slower. Compared to Fengshui5.0, mPL5 has 5% shorter wirelength and is 2X faster. The fast mode of mPL5 (mPL5-fast) can produce 1% shorter wirelength than FastPlace1.0 and is only 2X slower in average. A scalability plot of FastPlace1.0 and mPL5-fast is shown in Figure 5. It shows that mPL5-fast is slightly more scalable and it is expected that mPL5-fast will be faster than FastPlace1.0 on design with millions of cells. Figure 4 shows the average performance of each placer. The wirelength and runtime shown are divided by mPL5's wirelength and runtime respectively. We remark that we do not compare with Dragon's fixed-die mode (routability congestion driven) results as in [29], as it uses longer runtime and wirelength than the results of the default mode (wirelength driven). Also we compare to the latest binary of FastPlace1.0, which produces 5% shorter wirelength than the results published in [29] with similar runtime.

We also run mPL5 on PEKO examples [13] which are a set of synthetic benchmarks used to evaluate how far placement tools are from optimal. In [13], it shows that the quality of the current state-of-the-art placers is 50% to 150% from optimal. The results of mPL5 on PEKO suiteIII, circuits with pad connections, as well as other placers' results are shown in Figure 6. mPL5 can produce placement solution that is

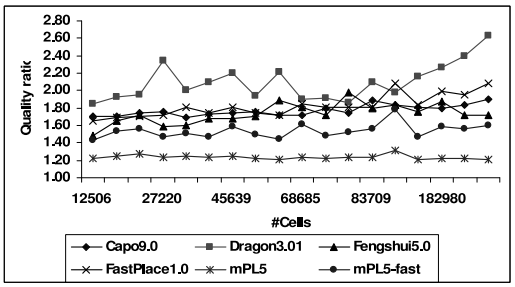

Figure 6: Quality ratio on PEKO suiteIII.

very close to the optimal—only 20% away, which again is the best among all existing placers.

5. CONCLUSION AND FUTURE WORK

To conclude, we have developed a multilevel generalized force-directed placement algorithm, named mPL5. Experiments show that mPL5 is a fast placement algorithm producing the shortest wirelength among the state-of-the-art academic placers. We remark that the GFD algorithm presented in the paper is not limited to standard cell placement. It is a general algorithm that can handle different objectives and density constraints. In the future, we will conduct experiments on mixed-block placement and add in supports of other constraints such as routability and thermal.

6. ACKNOWLEDGMENT

The authors would like to thank to Joseph Shinnerl and Min Xie for valuable discussions; Min Xie for the implementation of the detailed placement; Chris Chu and Natarajan Viswanathan for providing the benchmarks; Janice Martin-Wheeler and Jeanette Miller for the proofreading of the paper.

7. REFERENCES

[1] http://vlsicad.eecs.umich.edu/BK/ISPD02bench/.

[2] C. J. Alpert, D. J. Huang, and A. B. Kahng. Multilevel Circuit Partitioning. In *Proc. ACM/IEEE Design Automation Conference*, June 1997, pp. 627-632.

[3] C. R. Anderson and C. Elion. Accelerated Solutions of Nonlinear Equations Using Stabilized Runge-Kutta Methods, *UCLA CAM report*, April 2004.

[4] K. Arrow, L. Huriwicz and H. Uzawa. Studies in Nonlinear Programming. *Stanford University Press*, Stanford, CA, 1958.

[5] A. Brandt. Multiscale Scientific Computation: Review 2001. In T. Barth, R. Haimes, and T. Chan, editors, *Multiscale and Multiresolution Methods*. Springer Verlag, 2001.

[6] A. Brandt and D. Ron. Multigrid Solvers and Multilevel Optimization Strategies, chapter 1 of *Multilevel Optimization and VLSICAD*, Kluwer Academic Publishers, Boston, 2002.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can Recursive Bisection Alone Produce Routable Placements? In *Proc. Design Automation Conf.,* pp. 477-482, 2000.

[8] R. Chan, T. Chan, M. K. Ng, and A. Yip. Cosine Transform Preconditioner for High Resolution Image Reconstruction. *Linear Algebra Appls*, 316 (2000), pp. 89-104.

[9] T. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel Circuit Placement, chapter 4 of *Multilevel Optimization and VLSICAD*, Kluwer Academic Publishers, Boston, 2002.

[10] L. C. Evans. *Partial Differential Equations*, American Mathematical Society, 2002.

[11] T. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel Optimization for Large-scale Circuit Placement. In *Proc. IEEE International Conference on Computer Aided Design*, pp. 171-176, San Jose, CA, Nov 2000.

[12] T. Chan, J. Cong, J. Shinnerl, T. Kong and K. Sze. An Enhanced Multilevel Algorithm for Circuit Placement. In *Proc. IEEE International Conference on Computer Aided Design*, pp. 299-306, San Jose, CA, Nov 2003.

[13] C-C. Chang, J. Cong, and M. Xie. Optimality and Scalability of Existing Placement Algorithms. In *Proc. Asia South Pacific Design Automation Conference*, pp. 621-627, 2003.

[14] C-C. Chang, J. Cong, and X. Yuan. Multi-level Placement for Large-scale Mixed-size IC Designs.. In *Proc. Asia South Pacific Design Automation Conference*, pp.325-330, 2003.

[15] J. Cong. An Interconnect-Centric Design Flow for Nanometer Technologies. In *Proc. of the IEEE, vol. 89, No. 4*, pp. 505-528, April 2001.

[16] H. Eisenmann and F. M. Johannes. Generic Global Placement and Floorplanning. In *Proc. 35th ACM/IEEE Design Automation Conference*, pp. 269-274, 1998.

[17] S. W. Hur and J. Lillis. Mongrel: Hybrid techniques for standard-cell placement. In *Proc. IEEE International Conference on Computer Aided Design*, pp. 165-170, San Jose, CA, Nov 2000.

[18] A. B. Kahng and Q. Wang. Implementation and Extensibility of an Analytic Placer. In *Proc. International Symposium on Physical Design*, pp. 18-25, 2004.

[19] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526-529, 1997.

[20] A. A. Kennings and I. L. Markov. Analytical Minimization of Half-Perimeter Wirelength. In *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, pp. 179-184, Jan 2000.

[21] 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, March 1991.

[22] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. Analytical Placement: A Linear or a Quadratic Objective Function?. In *Proc. 28th ACM/IEEE Design Automation Conference* pp. 427-432, 1991.

[23] C. Li and C.-K. Koh. *On Improving Recursive Bipartitioning-based Placement*. Technical Report TR-ECE-03-14, Purdue University ECE, 2003.

[24] I. I. Mahmoud, K. Asakura, T. Nishibu and T. Ohtsuki. Experimental Appraisal of Linear and Quadratic Objective Functions Effect on Force Directed Method for Analog Placement. In *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences* 4(E77-A), pp. 710-725, 1994.

[25] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*, Cambridge University Press, 1994.

[26] W. Naylor et al.. *Non-linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer*. US Patent 6301693, Oct. 2001.

[27] Takuya Ooura. A FFT Package, http://momonga.t.u-tokyo.ac.jp/~ ooura/fft.html.

[28] L. I. Rudin, S. J. Osher and E. Fatermi. Nonlinear Total Variation Based Noise Removal Algorithms. In *Physica D 60* (1992), pp. 259-268.

[29] Natarajan Viswanathan and Chris Chong-Nuen Chu. FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model. In *Proc. International Symposium on Physical Design*, pp. 26-33, 2004.

[30] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell Placement Tool for Large Industry Circuits. In *Proc. Int. Conf. on Computer Aided Design*, pp. 260-276, 2000.

[31] M. C. Yildiz and P. H. Madden. Improved Cut Sequences for Partitioning Based Placement. In *Proc. ACM/IEE Design Automation Conf.*, 2001, pp. 776-779.

[32] J. Vygen. Algorithms for Large-scale Flat Placement. In *Proc. ACM/IEEE Design Automation Conf.*, pp. 746-751, 1997.

[33] Z. Xiu, J. D. Ma, S. M. Fowler, and R. A. Rutenbar. Large-Scale Placement by Grid-Warping. In *Proc. of IEEE/ACM Design Automation Conf.*, Jun. 2004.