Implementation and Extensibility of an Analytic Placer *

Andrew B. Kahng and Qinke Wang

UCSD CSE Department
La Jolla, CA 92093-0114 USA
{abk,qiwang}@cs.ucsd.edu

ABSTRACT

Automated cell placement is a critical problem in VLSI physical design. New analytical placement methods that simultaneously spread cells and optimize wirelength have recently received much attention from both academia and industry. A novel and simple objective function for spreading cells over the placement area is described in the patent of Naylor et al. [26]. When combined with a wirelength objective function, this allows efficient simultaneous cell spreading and wirelength optimization using nonlinear optimization techniques. In this work, we implement an analytic placer (APlace) according to these ideas (which have other precedents in the open literature), and conduct in-depth analysis of characteristics and extensibility of the placer. Our contributions are as follows. (1) We perform analysis and empirical studies of relevant characteristics of the objective functions described in [26]. (2) We extend the objective functions with congestion information. (3) We implement a top-down hierarchical (multilevel) placer (APlace) based on the objective functions. The half-perimeter wirelength of APlace outperforms that of Cadence QPlace (SE5.4), UCLA Dragon (v3.01) and Capo (v8.7) respectively by 6.8%, 2.6% and 6.5% on average. When these placements are detail-routed using Cadence WRoute (SE5.4), the average improvement in final wirelength is 8.2%, 4.2% and 10.4% over QPlace, Dragon and Capo, respectively. (4) We extend the placer to perform I/O-core co-placement. I/Os can be evenly distributed without damaging the wirelength figure of merit. (5) We also extend the placer to handle constraints for mixed-signal designs (symmetry, alignment, etc.) and evaluate the impact of such constraints on runtime and wirelength.

*Work partially supported by Cadence Design Systems, Inc., the California MICRO program, the MARCO Gigascale Silicon Research Center, NSF MIP-9987678 and the Semiconductor Research Corporation.

ISPD'04, April 18–21, 2004, Phoenix, Arizona, USA.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*; J.6 [**Computer Applications**]: COMPUTER-AIDED ENGINEERING

General Terms

Algorithms, Design, Performance

Keywords

Analytical Placement, Hierarchical Placement, Congestion, I/O-Core Co-Placement, Geometric Constraints

1. INTRODUCTION

Automated cell placement is a critical problem in VLSI design. As deep-submicron technology scales, new challenges arise for placement tools: design sizes are larger, turnaround times are shorter, and a variety of additional physical and geometrical constraints must be fulfilled simultaneously.

A common placement formulation seeks to minimize wirelength under the constraint that cells do not overlap each other. The current state-of-the-art placement tools can be classified into two categories, based on how they obtain a placement without cell overlaps [9]. The first class consists of algorithms that keep the placement free of overlaps during the whole placement process. For example, TimberWolf [29] is a well-known annealing based placement tool; it develops new placements by permuting an existing placement. The second class of algorithms is based on a hierarchical subdivision of the placement area with a corresponding partitioning of cells. Within this approach, a min-cut objective [2] [18] has been successfully used. A number of placement tools fall into this category: recursive partition with min-cut objective [4] [14] [31] [35] (Capo, CPlace, FengShui), quadratic placement [21] [30] [32] (GORDIAN, PROUD), and analytic placement with linear wirelength [28] (GORDIAN-L). Recently, the Dragon [33] [34] placement tool was presented, combining annealing with recursive bisection. Constructive methods based on (hybrids of) partitioning and analytical techniques are usually fast and produce good results. However, wirelength minimization may come at the cost of routability, or the inability to handle hard constraints.

New analytical placement methods that simultaneously spread cells and optimize wirelength have recently received much attention from both academia and industry. In such methods, forces based on the current cell distribution are applied to reduce cell overlaps. A quadratic objective combining wirelength and additional forces is proposed by [9]; in each iteration, a better spreading of cells is achieved without excessive net stretching. Another model of cell attracting and repelling (ARP) is presented in [10] [11]. Attractors (dummy cells) are added to sparse regions to drag cells from nearby dense regions, together with a cell repeller model that captures the wirelength objective. The ideas of additional forces and fixed pseudocells are also combined in [15]. The difficulty of force and attractor-directed placement methods is that wirelength is easily damaged by improper forces and attractors.

A novel and simple objective function for spreading cells over the placement area was proposed in the recent patent of Naylor et al. [26]. Combined with a wirelength objective function, it allows efficient simultaneous cell spreading and wirelength optimization using nonlinear optimization techniques. The focus of our present work is the implementation of an analytic placer (APlace) according to this idea, followed by in-depth analysis of characteristics and extensibility of the placer. While we implement the placer relying largely on the description in [26], it should be noted that this method integrates quite a few ideas that have been published in the open literature, as we describe in detail in Section 2 below. The main contributions of our work include the following.

- We perform analysis and empirical studies of relevant characteristics of the objective functions described in [26].
- We extend the objective functions with congestion information, to improve routability of placement results.
- We implement a top-down hierarchical (multilevel) placer (APlace) based on the objective functions. The half-perimeter wirelength of APlace outperforms that of Cadence QPlace (SE5.4), UCLA Dragon (v3.01) and Capo (v8.7) respectively by 6.8%, 2.6% and 6.5% on average. When these placements are detail-routed using Cadence WRoute (SE5.4), the average improvement in final wirelength is 8.2% over QPlace, and 4.2% over Dragon, and 10.4% over Capo.
- We extend the placer to perform I/O-core co-placement for area-array I/O designs. I/Os can be evenly distributed without damaging the wirelength figure of merit.
- We also extend the placer with constraint handling for mixed-signal designs. Basic geometric constraints including alignment, spacing and symmetry constraints can be enforced during placement process.

The remainder of this paper is organized as follows. Section 2 discusses the objective functions of the placer. Section 3 describes our implementation, and Section 4 summarizes the placement results. In Sections 5 and 6, the placer is extended with I/O-core co-placement and constraint handling. The paper concludes in Section 7.

2. PROBLEM FORMULATION

In this section, we discuss basic objective functions that our placer uses to distribute cells and minimize wirelength. Experiments are performed to study characteristics of the objectives.

	$r = 2$			$r = 4$		
grids	disc	iters	CPU(s)	disc	iters	CPU(s)
10	1.656	50	56	1.890	183	399
20	1.159	29	30	2.799	113	237
30	1.121	46	47	1.162	46	108
40	1.125	83	81	1.089	42	92
50	1.079	235	206	1.128	54	116
60	1.063	758	695	1.111	56	123
70	-	-	-	1.077	83	190
80	-	-	-	1.083	111	235
90	-	-	-	1.083	167	343
100	-	-	-	1.081	273	577

Table 1: Cell distribution results with different number of grids and cell potential radii (r's) for the ibm01-easy circuit.

2.1 Cell Spreading

One important objective of a placer is to distribute cells evenly over the placement area. Constructive placement methods can easily achieve this goal. Force-directed placement methods apply forces based on the current area distribution to move cells away from high-density regions and toward low-density regions. However, it is difficult to choose appropriate forces; wirelength is often seriously damaged when the cells are spread out.

A generic strategy divides the placement area into grids and seeks to equalize the total cell area in every grid. The penalty for an uneven cell distribution is:

$$Penalty = \sum_{Grid\ g} (TotalCellArea(g) - AverageCellArea)^2 \tag{1}$$

However, this penalty function is not smooth or differentiable, and is hence difficult to optimize. Naylor et al. [26] tried to smooth the above penalty function, proposing a bell-shaped cell potential function instead of a solid cell area function. For a cell c with center at $(CellX, CellY)$ with area A, the potential at grid point $g = (GridX, GridY)$ is:

$$\begin{aligned} &Potential(c, g) = \\ &\alpha(c) \cdot p(|CellX - GridX|) \cdot p(|CellY - GridY|) \end{aligned} \tag{2}$$

where

$$p(d) = \begin{cases} 1 - 2d^2/r^2 & (0 \le d \le r/2) \\ 2(d-r)^2/r^2 & (r/2 \le d \le r) \end{cases} \tag{3}$$

Here, r controls the *radius* of the cells' potential and $\alpha(c)$ is a proportionality factor so that $\sum_g Potential(c, g) = A$. Then, the penalty function in Equation (1) is transformed to:

$$Penalty = \sum_{Grid\ g} (\sum_{Cell\ c} Potential(c, g) - ExpPotential(g))^2 \tag{4}$$

where $ExpPotential(g) = TotalCellArea/NumGrids$ is the expected total potential at the grid point g.

We can use the conjugate gradient method to minimize the penalty function in Equation (4). Our implementation will be described in detail in Section 3. We calibrate our optimizer using the ibm01-easy testcase from the IBM-PLACE 2.0 benchmark suite [16]. Our stopping criterion is when the maximum (Manhattan) movement of any cell between consecutive iterations is less than 30 units. Results with different numbers of grids and cell potential radii (r's) are summarized in Table 1.

grids	α	init WL	final HPWL	CPU (s)
10	3336	7.533	0.803	194
20	1668	7.369	0.836	262
30	1112	7.337	0.913	99
40	834	7.326	1.274	238
50	667	7.321	1.400	245
60	556	7.318	1.499	245
70	476	7.316	1.583	252
80	417	7.315	1.653	266
90	370	7.314	1.712	240
100	334	7.314	1.764	248

Initial HPWL = 7.311

Table 2: Wirelength minimization results with different smoothing parameters (α's) for the ibm01-easy circuit.

We use *discrepancy* to measure evenness of cell distribution. *Discrepancy within area A* is defined as the maximum ratio of actual total cell area to expected cell area over all windows with area A. In Table 1, the second and fifth columns show the discrepancy within 1% of the total placement area, for each of the cell distributions that the placer obtains. The number of iterations, and total running times needed to meet the above convergence criterion, are also shown in the table.

From these tests, we make the following conclusions. (1) The finer the grid, the more iterations needed for the optimizer to converge; however, a more even cell distribution is obtained. (2) For finer grids, larger values of r help to reduce the number of iterations, although the running time per iteration is increased. (3) We observe serious oscillations in discrepancy when $r = 2$ and the number of grids is larger than 60.

2.2 Wirelength Formulation

Minimization of wirelength is a common objective for circuit placement. Linear and quadratic wirelength objectives are typically used; see, e.g., [24] and [28] for comparisons. The quadratic objective function is used in many analytical placement methods because it is continuously differentiable and can be minimized efficiently by solving a system of linear equations. Unfortunately, this is not true for linear objective functions, and linear programming suffers from excessive computation times. The Gordian-L objective [28] minimizes a linear wirelength function using quadratic programming methods. Also, [23] proposes an α-order objective function to capture the strengths of both methods.

Placement quality is typically evaluated in terms of half-perimeter wirelength (HPWL). Convex nonlinear approximations of HPWL, which do not require net models and which permit direct inclusion of nonlinear delay terms, are proposed and well-studied in [1] [3] [19] [20]. The approach of Naylor et al. [26] similarly uses a log-sum-exp method to capture the linear half-perimeter wirelength while simultaneously obtaining the desirable characteristic of continuous differentiability. For a net t with pin coordinates $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$, the HPWL is formulated as:

$$\begin{aligned} WL(t) = \quad & \alpha \cdot (ln(\sum e^{x_i/\alpha}) + ln(\sum e^{-x_i/\alpha})) + \\ & \alpha \cdot (ln(\sum e^{y_i/\alpha}) + ln(\sum e^{-y_i/\alpha})) \end{aligned} \tag{5}$$

where α is a smoothing parameter.[1] The log-sum-exp formula picks the most dominant terms[2]; it has been previously used in physical design applications such as transistor sizing [27].

We minimize the wirelength objective function using the conjugate gradient optimizer. Initially, cells are randomly distributed over the placement area, and then the wirelength objective function in Equation (5) is minimized. The program is stopped after 300 iterations. Results with different smoothing parameter values (α's) are shown in Table 2. The wirelength calculated according to Equation (5) for the initial random placement is displayed in the third column. Compared to the actual initial HPWL of this random placement (i.e., Initial HPWL = 7.311), we see that the wirelength formulation becomes more accurate when smaller values of α are used. However, with larger α values, the wirelength objective function can be minimized more quickly, and a smaller final HPWL is obtained (e.g., final HPWL = 0.803 for $\alpha = 3336$).

Combining the above two objectives, the analytic placer optimizes the following function:

$$WLWeight * TotalWL + DensityWeight * DensityPenalty \tag{6}$$

The density term drives the spreading of cells and is always changing with the current cell distribution. The wirelength term draws connected components back toward each other.

3. DETAILED IMPLEMENTATION

We now describe implementation details of APlace.

3.1 Conjugate Gradient Optimizer

We use the conjugate gradient method to optimize the objective function as described in Equation (6). The conjugate gradient method is quite useful in finding an unconstrained minimum of a high-dimensional function $f(x)$. A detailed treatment, along with a survey of descent-based methods for nonlinear programming, can be found in [25].

In general, the conjugate gradient method finds the minimum by executing a series of *line minimizations* (i.e., line searches). A line minimization corresponds to one-dimensional function minimization along some search direction. The result of one line minimization is used as the start point for the next line minimization. The method has the following form:

$$d_k = \begin{cases} -g_k & for\ k = 1 \\ -g_k + \beta_k d_{k-1} & for\ k > 1 \end{cases} \tag{7}$$

$$x_{k+1} = x_k + \alpha_k d_k \tag{8}$$

where g_k denotes the gradient $\nabla f(x_k)$, α_k is a step length obtained by a line search algorithm, d_k is the search direction, and β_k is chosen so that d_k becomes the k^{th} conjugate direction when the function is quadratic and the line search finds the minimum along the direction exactly. Varieties of conjugate gradient methods differ in how they select β_k: the best-known formulas for β_k are due to Fletcher-Reeves, Polak-Ribiere, and Hestenes-Stiefel. Our implementation uses the Polak-Ribiere formula:

$$\beta_k^{PR} = \frac{g_k^T(g_k - g_{k-1})}{||g_{k-1}||^2}. \tag{9}$$

[1] A similar "smoothening" parameter is used in the wirelength formulation of Kennings and Markov [19].

[2] Compared to the function in [19], this wirelength formulation is harder to compute as it uses transcendental functions.

Top-Down Hierarchical APlace Algorithm

Input:

User-defined objective density discrepancy $DestDisc$
User-defined minimum step length ϵ
User-defined total number of iterations $TotalIters$

Output:

Cell placement $\{CellPosition(c)\}$

Algorithm:

Construct a hierarchy of clusters $\{C_l(i)\}$
For $(l = MaxLevel;\ l >= 0;\ l = l - 1)$
    Set initial placement $\{ClusterPosition_l(i)\}$
    $GridLen \propto \sum_i sqrt(Area(C_l(i)))/N_l$
    $alpha \propto GridLen$
    $DensityWeight = 1$
    $WLWeight \propto \frac{DensityPenalty(GridLen,\ r)}{Wirelength(\alpha)}$
    While $(\ \#Iter < TotalIters)$
        $f = DensityPenalty(gridLen,\ r) +$
            $WLWeight * Wirelength(\alpha)$
        $Gradient_l = Gradient(f)$
        $StepLength = LineSearch(f, Gradient_l)$
        $ClusterPosition_l = CellPosition_l +$
            $StepLength * Gradient_l$
        If $(StepLength < \epsilon)$
            $WLWeight = WLWeight/2$
        If $(Discrepancy(gridLen) > DestDisc)$
            $Break$
$\{CellPosition(c)\} = \{ClusterPosition_0(i)\}$

Figure 1: Top-Down Hierarchical APlace

A Golden Section search method is used to find the step length for each iteration. The length of the search interval reduces by a factor of 0.618 (the golden ratio) or more in each step, and converges linearly to zero.

The conjugate gradient iteration in Equation (8) repeats until the following stopping criterion is reached: (1) a predetermined number of iterations has passed; or (2) the step length returned by the line search function is small enough; or (3) the function value is not changing significantly with additional iterations.

3.2 Control Factors

The weights of the wirelength and density objective functions provide important control parameters to the placer. A larger weight for the wirelength objective contracts cells together and prevents them from spreading out. A larger weight for the density penalty spreads the cells out hastily without a good wirelength. In APlace, the density weight is fixed at a constant, and the wirelength weight is set to be large in the beginning. When the conjugate gradient optimizer slows down and a stable solution appears, the wirelength weight is divided by 2 (or a smaller number when the placer is near its final stages). The conjugate gradient optimizer then computes a new stable state, wherein cells are distributed more evenly but wirelength is larger. The process repeats until the cells are spread evenly over the placement area.

The number of grids, cell potential parameter r and wirelength smoothing parameter α are also important control knobs for APlace. According to the empirical studies discussed in Section 2 above, the number of grids should increase during the whole placement process. Coarser grids at the beginning spread out the cells faster, while finer grids at the final stages help to reach a more even distribution. In our implementation, the cell potential radius r is set to 2 for coarser grids in order to reduce running times, and to 4 for finer grids.[3] The wirelength smoothing parameter α is kept proportional to the grid length.[4]

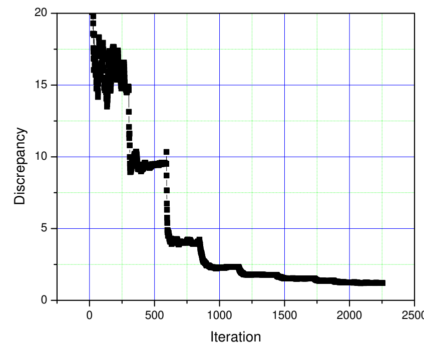

Figure 2: Discrepancy as a function of iterations for the ibm01-easy circuit.

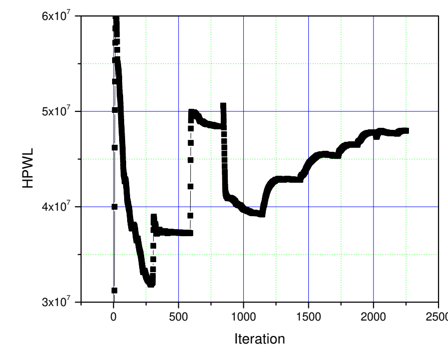

Figure 3: Half-perimeter wirelength as a function of iterations for the ibm01-easy circuit.

3.3 Top-Down Hierarchical Algorithm

We use a top-down hierarchical approach to accelerate APlace. During initialization, a hierarchy of clusters is constructed using MLPart 4.21, a leading-edge, open-source min-cut hypergraph/circuit partitioner [5]. The top-down hierarchical algorithm is described in Figure 1. Notations used are summarized as follows:

n	number of cells
$MaxLevel$	maximum number of cluster levels
N_l	number of clusters at level l
$C_l(i)$	a cluster of cells at level l
$Area(C)$	total cell area of cluster C
$GridLen$	spacing of grids
f	objective function of the placer
$\{Gradient_l(i)\}$	gradient vector
$\{ClusterPosition_l(i)\}$	a vector of cluster positions
$\{CellPosition(c)\}$	a vector of cell positions

Subscript ranges, where not explicit, are: $c = 1,\ ...,\ n;\ l = 0,\ ...,\ maxLevel$; and $i = 1,\ ...,\ N_l$.

For each level in the cell/cluster hierarchy, a coarse grid is determined by the average cluster size. We compute the density penalty by regarding cells in a cluster as a macro

[3] In our experiments, r is set to 2 for most grids, and to 4 only for the finest grid.

[4] In our experiments, α is set to be half of the grid length.

γ	Placement				Global Routing	
	WL	WL⌟	disc	iters	WL	over-cap
0.00	0.459	0.502	1.18	1006	0.119	4035
0.02	0.462	0.505	1.18	997	0.118	3488
0.04	0.464	0.509	1.23	1006	0.119	3249
0.05	0.474	0.523	1.26	1086	0.120	2486
0.06	0.477	0.529	1.28	1086	0.121	2576
0.08	0.486	0.541	1.36	1006	0.123	2806
0.10	0.507	0.562	1.56	804	0.129	3350

Table 3: Placement and global routing results of APlace with varying congestion adjustment factors (γ's) for the ibm01-hard circuit.

cell with area equal to the total cell area of the cluster. For wirelength calculation, cells are assumed to be located at the center of the cluster.

Figures 2 and 3 show how discrepancy and HPWL change with successive iterations for the ibm01-easy circuit. The clustering hierarchy for ibm01-easy has three levels. During the first approximately 900 iterations, the placer is working at cluster levels. Clustering helps to spread cells more quickly, but wirelength is impaired during cell expansion. It is clearly seen from the figures that when wirelength weight is decreased and the conjugate gradient optimizer restarts, discrepancy drops sharply and wirelength is often increased at first and then refined during the optimization. When both discrepancy and wirelength change slowly, the wirelength weight should be reduced. Guided by these figures, we set the iteration limit at 100 in our experiments below.

3.4 Legalization

The placement results of APlace have cell overlaps and need to be legalized. A simplified Tetris [13] legalization algorithm is implemented in APlace. After the global placement, cells are sorted according to their vertical coordinates, and then from left to right, we search the current nearest available position for each cell. This simple algorithm is fast (running time is ignorable compared to that of global placement) and increases wirelength by about 5% for IBM-PLACE 2.0 circuits [16].

3.5 Congestion-Directed Placement

To improve routability of placement results, we have integrated congestion information into the objective functions to direct cell distribution. We use Kahng and Xu's accurate bend-based congestion estimation method [17] in our placer. If a particular grid is determined to be congested, the expected total cell potential of the grid in Equation (4) is reduced accordingly, and vice versa. The sum of expected area potential over all grids is kept constant, and equal to the total cell area. The formula for adjusting expected cell potential is as follows:

$$ExpPotential(g) \propto 1+\gamma(1-2\frac{Congestion(g)}{\max_g\{Congestion(g)\}}) \quad (10)$$

where γ is the congestion adjustment factor and decides the extent of congestion-directed placement.

Table 3 summarizes placement runs for the ibm01-easy circuit with different congestion factors (γ's). Specific parameters used for these runs are: $\epsilon = 30$ and $destDisc = 1.1$. Wirelengths before and after legalization are shown in the second and third columns. Discrepancy within 1% placement area is shown in the fourth column, and serves as a measure of evenness of cell distribution. The fifth column shows the total number of iterations. Global routing is performed using Cadence WRoute (SE5.4). The last two columns show the total wire length in the gcell grid, and total number of over-capacity gcells; these values provide figures of merit for routability of the placement results.

IBM-PLACE 2.0			APlace				
ckts	cells	nets	WL	WL⌟	disc	iters	CPU
ibm01e	12282	11507	0.48	0.52	1.19	1098	13
ibm01h	12028	11507	0.46	0.50	1.18	1006	21
ibm02e	19321	18429	1.41	1.45	1.12	1097	30
ibm02h	19062	18429	1.38	1.44	1.11	1208	32
ibm07e	45135	44394	3.17	3.29	1.14	968	64
ibm07h	44811	44394	3.09	3.24	1.15	968	51
ibm08e	50977	47944	3.51	3.65	1.11	887	75
ibm08h	50672	47944	3.45	3.68	1.11	806	55

Table 4: Placement results of APlace (no congestion awareness) for eight IBM-PLACE 2.0 circuits.

According to these results, routability is approximately 38% better with a congestion factor of 0.05, but deteriorates when the congestion factor is larger. Wirelength of congestion-aware placements is not seriously impaired.

4. EXPERIMENTAL RESULTS

APlace is implemented in C++. We pick eight circuits from the IBM-PLACE 2.0 benchmark suites [16] as our test cases. The eight circuits include easy and hard cases of four designs. IBM-PLACE circuits are widely used in the literature, and are becoming common benchmarks for standard-cell placement.

Our results are compared with a leading industry placer, Cadence QPlace (SE5.4), and recent academic tools: UCLA Dragon (v3.01) [8] and Capo (v8.7) [7]. All placers read the same LEF/DEF files and write DEF files as the placement output. Results of Capo are legalized using Cadence QPlace (SE5.4, ECO mode[5]). The placements are then sent to Cadence WRoute (SE5.4) to be routed.[6]

All the experiments of APlace, Dragon and Capo are performed on a Xeon server with 2.4GHz CPU and 4GB memory; QPlace and WRoute are run on a Sun Ultra10 workstation with 400MHz CPU. We run QPlace with full placement mode (-full), and Dragon with fixed die mode (-fd). We run MetaPlacer for Linux, which incorporates Capo and orientation optimizer. Row ironing is disabled (-noRowIroning) for MetaPlacer for better routability of the placement results.

The APlace results with no congestion awareness are summarized in Table 4. Specific parameters used for these runs are: $\epsilon = 30$ and $destDisc = 1.1$. Wirelengths (meters) before and after legalization are shown in the fourth and fifth columns. Average wirelength increase after legalization is 5.7% (range: 3.4% to 9.4%). Better wirelength can be achieved using other legalization tools.[7] Discrepancy within 1% placement area is shown in the sixth column and used to evaluate the evenness of cell distribution in APlace results. The last two columns show the total number of iterations and running times in minutes.

[5] QPlace is run with ECO placement enabled and other options in default modes.

[6] WRoute is run with global and final routing enabled, autostop disabled and other options in default modes.

[7] Preliminary experiments show a legalized wirelength of 1.36 for the ibm02-easy circuit and 3.25 for the ibm08-easy circuit using the legalizer of [22].

Ckts	Placer	Place		Route			
		WL	CPU	viol	WL	vias	CPU
ibm01e	QPlace	0.59	3	0	0.84	138563	58
	Dragon	0.57	27	0	0.86	141304	60
	Capo	0.57	1	587	0.85	146706	1446*
	APlace	**0.55**	**23**	**0**	**0.80**	**152489**	**195**
ibm01h	QPlace	0.56	3	0	0.80	138593	82
	Dragon	0.55	26	0	0.80	139993	90
	Capo	0.56	1	1029	0.84	173715	1446*
	APlace	**0.52**	**22**	**0**	**0.75**	**150947**	**253**
ibm02e	QPlace	1.56	7	0	2.10	291154	58
	Dragon	1.60	38	0	2.20	310679	243
	Capo	1.60	2	35	2.33	317659	636
	APlace	**1.52**	**37**	**0**	**2.05**	**299306**	**48**
ibm02h	QPlace	1.52	7	0	2.18	305420	104
	Dragon	1.47	40	0	2.16	304376	105
	Capo	1.56	2	147	2.21	319702	929
	APlace	**1.50**	**38**	**0**	**2.14**	**315786**	**176**
ibm07e	QPlace	3.72	12	0	4.61	572512	98
	Dragon	3.66	65	0	4.58	569087	103
	Capo	3.71	7	42	4.93	599806	996
	APlace	**3.37**	**66**	**0**	**4.18**	**559354**	**95**
ibm07h	QPlace	3.70	12	0	5.04	617942	184
	Dragon	3.44	66	15	4.63	606561	135
	Capo	3.56	8	1799	5.14	631456	1483*
	APlace	**3.32**	**54**	**1**	**4.29**	**586129**	**238**
ibm08e	QPlace	3.95	16	0	5.30	703891	133
	Dragon	3.61	143	0	4.58	659802	57
	Capo	3.93	7	0	5.07	682815	137
	APlace	**3.64**	**78**	**0**	**4.58**	**681884**	**107**
ibm08h	QPlace	3.85	17	0	5.26	726576	194
	Dragon	3.45	143	0	4.47	679761	100
	Capo	3.90	8	1484	5.15	728195	1458*
	APlace	**3.56**	**48**	**0**	**4.63**	**699411**	**165**

* WRoute reaches time limit (24h).

Table 5: Placement and routing results of APlace (with congestion awareness) for eight IBM-PLACE 2.0 circuits with comparison to QPlace, Dragon and Capo.

The placement results with congestion awareness are sent to Cadence WRoute (SE5.4) to be routed. The results are compared to QPlace, Dragon and Capo in Table 5. Routing results include *success* (finished routing with no violations), *finished* (with violations) and *failure* (because of time limit). For all cases, the number of violations, final wirelength, the number of vias and running times of WRoute are shown in the last four columns.

According to the results, for placements with no congestion awareness, average *placed* wirelength improvement over QPlace is 9.0% (range: 4.5% to 12.7%); average improvement over Dragon is 4.8% (range: -6.5% to 10.2%); average improvement over Capo is 8.7% (range: 5.7% to 11.4%). For the placement results with congestion awareness, average *placed* wirelength improvement over QPlace is 6.8% (range: 1.9% to 10.1%); average improvement over Dragon is 2.6% (range: -3.2% to 7.9%); average improvement over Capo is 6.5% (range: 4.1% to 9.1%). Running times of APlace increase roughly linearly with the number of cells and nets. APlace is faster (0.8X) than Dragon, but much (13.2X) slower than Capo.

We observe that almost all of the circuits are successfully routable with good wirelength; finished routings with a small number of violations can be manually fixed. Average improvement of the *routed* wirelength over QPlace is 8.2% (range: 1.8% to 14.8%); average improvement over Dragon is 4.2% (range: -3.6% to 8.7%); average improvement over Capo is 10.4% (range: 3.1% to 16.6%). The routability of APlace's results is better than that of Capo's; routed wirelength of APlace outperforms that of QPlace and Dragon with an average of less than 2% increase in the number of vias and less than 80% increase in running time of WRoute.

IBM-PLACE		APlace with IO-Core Co-Placement					
ckts	I/Os	WL	WL_l	I/O disc	disc	iters	CPU (m)
ibm01e	0	0.48	0.52	-	1.19	1098	13
	400	0.48	0.52	1.50	1.24	1144	25
	1000	0.50	0.54	1.34	1.31	1274	28
ibm02e	0	1.41	1.45	-	1.12	1097	30
	400	1.36	1.41	1.62	1.09	1107	30
	1000	1.45	1.50	1.36	1.16	1279	34
ibm07e	0	3.17	3.29	-	1.14	968	64
	400	3.36	3.47	1.70	1.14	1049	64
	1000	3.32	3.43	1.55	1.14	1049	59
ibm08e	0	3.51	3.65	-	1.11	887	75
	400	3.59	3.83	1.55	1.12	887	55
	1000	3.55	3.73	1.20	1.11	887	61

Table 6: I/O-core co-placement results of APlace with different number of I/Os.

5. I/O-CORE CO-PLACEMENT

IC packaging technologies with peripheral I/O pads have well-known shortcomings: clock/power distribution is constrained, and large parasitics of peripheral I/O pads cause coupling and power issues for off-chip signaling. The area-array I/O regime is projected to eventually dominate IC implementation methodology, affording improved pad count and reliability, and reduced noise coupling.

Area-array I/O presents new challenges to placement tools. Caldwell et al. [6] conducted a thorough study of the implication of area-array I/O for placement methods. They determined that with alternating I/O and core placement methods, which are often used in practice, the number of iterations needed to achieve good solutions can be surprisingly large. Also, a bad initial I/O placement can seriously handicap subsequent iterations. A simultaneous methodology is proposed in [12], which performs top-down hierarchical placement, with I/Os re-placed to each partition by min-cost assignment. However, previous results from the simultaneous methodology are not good.

I/Os and core cells can be simultaneously placed in APlace. I/Os are spread over the placement area, in the same way and at the same time as core cells. The objective function of APlace in Equation (6) is modified as:

$$WLWeight * TotalWL + DensityWeight * DensityPenalty \\ + IODensityWeight * IODensityPenalty \tag{11}$$

where the third term drives the spreading of I/Os.

We have tested extensions of APlace to perform I/O-core co-placement. We randomly select 400 or 1000 cells from IBM-PLACE circuits, equalize the sizes of these cells to be the average of their sizes, and regard them as I/Os. Specific parameters used for the experiments are: $\epsilon = 30$ and $destDisc = 1.1$. Figure 4 shows the placement for the ibm01-easy circuit with 400 I/Os. Core cells and I/Os are displayed as yellow (grey) blocks and pink (dark) blocks, respectively. I/Os are distributed evenly over the placement area in the figure. The results are summarized in Table 6. The third and fourth columns show the wirelength before and after legalization, in meters. I/O distribution is evalu-

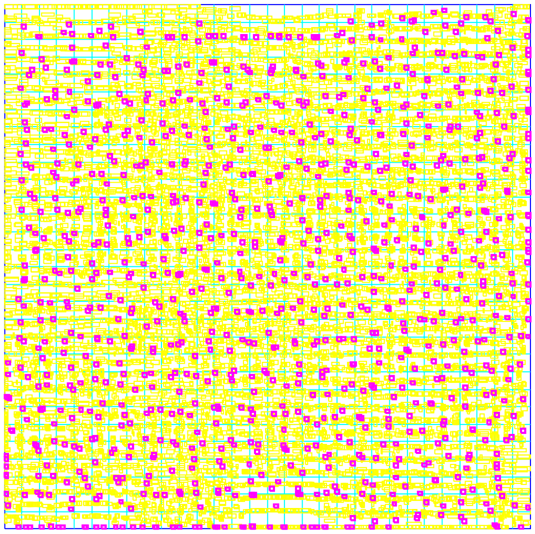

Figure 4: I/O-core co-placement of APlace with 400 I/Os for ibm01-easy circuit.

IBM-Dragon	APlace with Constraints				
Ckts	WL	WL_l	disc	iters	CPU (m)
ibm01e	0.54	0.57	1.17	1309	57
ibm02e	1.50	1.55	1.10	1208	70
ibm07e	3.36	3.48	1.16	1049	123
ibm08e	3.78	3.99	1.12	887	130

Table 7: Placement results of APlace with 90 artificial geometric constraints.

ated by the I/O discrepancy within 1% placement area. According to the results, I/Os are distributed fairly evenly over the placement area, especially when the number of I/Os is 1000. Wirelength, discrepancy of core cell distribution, and running time are not seriously impaired by the co-placement.

6. GEOMETRIC CONSTRAINTS

The need to increase the complexity and reduce the cost of electronic systems has greatly accelerated the demand for combining discrete components into application specific integrated circuits (ASICs). As more digital circuitry is integrated, the analog components of a system are more likely to represent a bottleneck in the path to size and cost reduction for a system. With increased demand for mixed-signal ASICs, there is a corresponding demand for software tools and design methodologies that increase the productivity of analog and mixed-signal ASIC designs.

Digitally targeted tools are often inadequate to handle the critical and specific requirements of analog layout. Performance of analog circuits is much more sensitive to the details of physical implementation than that of digital circuits. A large number of constraints have to be considered in order to avoid extra design iterations caused by too many parasitic effects. Layout synthesis is often a multi-objective optimization problem where, along with area, wiring length and delay, topological constraints must be taken into account.

Basic geometric constraints for mixed-signal placement include the following categories: fixed components, spacing, alignment, axial symmetry and nodal symmetry. These categories cover most of the topological requirements in an analog layout system. More complex constraints, such as

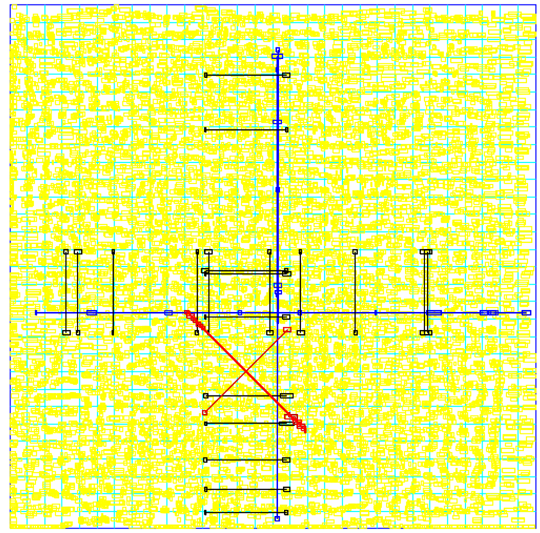

Figure 5: Placement of APlace with 90 artificial geometric constraints for ibm01-easy circuit.

matching, can be built as combinations of these basic constraints.

Geometric constraints can be handled directly in APlace, since they can be converted to penalty functions and added to the objective function of the placer. Penalty functions for basic geometric constraints are summarized as follows.

- For an x-alignment constraint for cells with coordinates $\{(x_1, y_1),\ (x_2, y_2),\ ...,\ (x_n, y_n)\}$, which is expressed as $x_1 = x_2 = \ ... \ = x_n$, the penalty function is $\sum (x_i - \bar{x})^2$.

- For an x-spacing constraint between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $L < |x_1 - x_2| < U$, the penalty function is
$$\begin{cases} (|x_1 - x_2| - L)^2 & \textit{when } |x_1 - x_2| < L \\ 0 & \textit{when } L < |x_1 - x_2| < U \\ (|x_1 - x_2| - U)^2 & \textit{when } |x_1 - x_2| > U \end{cases}$$

- For an axial symmetry constraint with an axis $y = c$ between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $x_1 = x_2$ *and* $y_1 + y_2 = 2c$, the penalty function is $(x_1 - x_2)^2 + (y_1 + y_2 - 2c)^2$.

- For a nodal symmetry constraint with a center (c_x, c_y) between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $x_1 + x_2 = 2c_x$ *and* $y_1 + y_2 = 2c_y$, the penalty function is $(x_1+x_2-2c_x)^2+(y_1+y_2-2c_y)^2$.

Constraint handling is implemented in APlace, and experiments are performed for each IBM-PLACE easy circuit with a manually specified combination of 40 spacing constraints, 20 alignment constraints, 20 axial symmetry and 10 nodal symmetry constraints. Specific parameters used for the experiments are: $\epsilon = 30$ and $destDisc = 1.1$. Figure 5 shows the placement with 90 geometric constraints for the ibm01-easy circuit. Constraints are shown as cells connected by colored (dark) lines. Cells connected by blue lines have alignment constraints among them; cells connected by black lines have axial symmetry and spacing constraints; and cells connected by red lines have nodal symmetry and spacing constraints. We can see from the figure that geometric constraints are closely enforced. The placement results are summarized in Table 7. The second and third columns show the wirelength before and after legalization in meters. Compared to placement results without constraints, the legalized wirelength is increased by 8.2% on average.

7. CONCLUSION AND FUTURE WORK

We have implemented APlace, an analytic placer based on ideas described in the recent patent of Naylor et al. [26], and have conducted in-depth analysis of characteristics and results of the placer. The implementation is successful: placed and routed wirelengths outperform QPlace and Capo, and are close to Dragon. We also extend the basic placer to perform top-down hierarchical placement, congestion-directed placement, I/O-core co-placement and constraint handling for mixed-signal contexts.

Our ongoing research directions include: (1) different wirelength objectives; and (2) extension to timing-driven placement.

8. REFERENCES

[1] C. J. Alpert, T. Chan, A. B. Kahng, I. Markov and P. Mulet, "Faster Minimization of Linear Wirelength for Global Placement", *IEEE Trans. Computer Aided Design* 17(1) (1998), pp. 3-13.

[2] C. J. Alpert, J.-H. Huang and A. B. Kahng, "Multilevel Circuit Partitioning", *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 530-533.

[3] R. Baldick, A. B. Kahng, A. Kennings and I. L. Markov, "Function Smoothing with Applications to VLSI Layout", *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, Jan. 1999, pp. 225-228.

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

[5] A. E. Caldwell, A. B. Kahng, A. A. Kennings and I. L. Markov, "Hypergraph Partitioning for VLSI CAD: Methodology for Reporting, and New Results", *Proc. ACM/IEEE Design Automation Conf.*, June 1999, pp. 349-354.

[6] A. E. Caldwell, A. B. Kahng, S. Mantik and I. L. Markov, "Implications of Area-Array I/O for Row-Based Placement Methodology", *Proc. IEEE Symp. IC/Package Design Integration*, Feb. 1998, pp. 93-98.

[7] UCLA/UMICH Physical Design Tools. `http://vlsicad.eecs.umich.edu/BK/PDtools/`.

[8] Web site of Dragon. `http://er.cs.ucla.edu/Dragon/`.

[9] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 269-274.

[10] H. Etawil, S. Areibi and A. Vannelli, "ARP: A Convex Optimization Based Method for Global Placement", submitted to *IEEE Trans. Computer Aided Design*, `http://wolfman.eos.uoguelph.ca/~sareibi1/PUBLICATIONS_dr/all-pub-list.html`.

[11] H. Etawil, S. Areibi and A. Vannelli, "Attractor-Repeller Approach For Global Placement", *Proc. IEEE/ACM int. Conf. Computer-Aided Design*, 1999, pp. 20-24.

[12] T. Gao, C. L. Liu and K. C. Chen, "A Performance Driven Hierarchical Partitioning Placement Algorithm", *Proc. European Design Automation Conf.*, Sep. 1993, pp. 33-38.

[13] D. Hill, "Method and System for High Speed Detailed Placement of Cells within an Integrated Circuit Design", *US Patent* 6370673, April 2002.

[14] D. J. Huang and A. B. Kahng, "Partitioning-Based Standard Cell Global Placement With an Exact Objective", *Proc. Int. Symp. Physical Design*, 1997, pp. 18-25.

[15] B. Hu and M. Marek-Sadowska, "FAR: Fixed-Points Addition & Relaxation Based Placement", *Proc. Int. Symp. Physical Design*, 2002, pp. 161-166.

[16] Web site of IBM-PLACE 2.0 benchmark suites. `http://er.cs.ucla.edu/benchmarks/ibm-place2/`

[17] A. B. Kahng and X. Xu, "Accurate Pseudo-Constructive Wirelength and Congestion Estimation", *Proc. ACM Int. Workshop on System-Level Interconnect Prediction*, April 2003, pp. 61-68.

[18] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Design", *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526-529.

[19] A. A. Kennings and I. L. Markov, "Smoothening Max-terms and Analytical Minimization of Half-Perimeter Wirelength", *VLSI Design* 14(3) (2002), pp. 229-237.

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

[21] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE Trans. Computer-Aided Design* 10(3) (1991), pp. 356-365.

[22] C.-K. Koh and C. Li, private communication, Oct. 2003.

[23] J.-M. Li, J. Lillis, L.-T. Liu and C.-K. Cheng, "New Spectral Linear Placement and Clustering Approach", *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 88-93.

[24] J.-M. Li, J. Lillis and C.-K. Cheng, "Linear Decomposition Algorithm for VLSI Design Applications", *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 223-228.

[25] K. G. Murty, *Linear Complementarity, Linear and Nonlinear Programming*, Internet Edition, Chapter 10, pp. 389-460. `http://ioe.engin.umich.edu/books/murty/linear_complementarity_webbook/`.

[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] S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-Driven Layout Synthesis*, Kluwer Academic Publishers, Boston, MA, 1993.

[28] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 427-431.

[29] W.-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits", *IEEE Trans. Computer-Aided Design* 14(3) (1995), pp. 349-359.

[30] R. S. Tsay, E. Kuh and C. P. Hsu, "PROUD: A Sea-Of-Gates Placement Algorithm", *IEEE Design & Test of Computers* (1988), pp. 44-56.

[31] P. Villarrubia, G. Nusbaum, R. Masleid and P. T. Patel, "IBM RISC Chip Design Methodology", *Proc. Int. Conf. Computer Design* , 1989, pp. 143-147.

[32] J. Vygen, "Algorithms for Large-Scale Flat Placement", *Proc. ACM/IEEE Design Automation Conf. '97*, pp. 746-751.

[33] X. Yang, B.-K. Choi and M. Sarrafzadeh, "A Standard-Cell Placement Tool for Designs with High Row Utilization", *Proc. Int. Conf. Computer Design*, Sep. 2002, pp. 45-47.

[34] X. Yang, B.-K. Choi and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement", *Proc. Int. Symp. Physical Design*, 2002, pp. 42-47.

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