# Fast Analytic Placement using Minimum Cost Flow

Ameya R. Agnihotri Patrick H. Madden
SUNY Binghamton Computer Science Department
Box 6000, Binghamton NY 13902
email: {ameya, pmadden}@acm.org

## ABSTRACT

Many current integrated circuits designs, such as those released for the ISPD2005[14] placement contest, are extremely large and can contain a great deal of white space. These new placement problems are challenging; analytic placers perform well, but can suffer from high run times. In this paper, we present a new placement tool called *Vaastu*. Our approach combines continuous and discrete optimization techniques. We utilize network flows, which incorporate the more realistic half-perimeter wire length objective, to facilitate module spreading in conjunction with a log-sum-exponential function based analytic approach. Our approach obtains wire length results that are competitive with the best known results, but with much lower run times.

## I. INTRODUCTION

Placement is one of the most critical stages during the design cycle of an integrated circuit. Modern integrated circuits can contain millions of logic modules. Producing a physical layout of these modules is an intimidating task. To ease the process, designers often use Intellectual Property (IP) modules; these may have fixed positions, and need to be treated as obstacles during placement. In addition to the fixed obstacles, there may also be movable macro blocks and millions of unplaced standard cells.

Many of these modern designs contain a large amount of white space, to support optimization for buffer insertion and gate sizing. The ISPD2005 benchmarks display these characteristics, with white space ranging from 27% to 57%. These designs, which are representative of current circuits, are quite challenging.

Various state-of-the-art placers exist today which can be categorized as below. We include placers with recently published work.

1. Analytic placers : APlace [11], BonnPlace [4] , FastPlace [20], hATP [15], Kraftwerk [16], mFAR [9] and mPL [5] are recent analytic placers. APlace and mPL6 are non-linear optimization approaches, where as, BonnPlace, FastPlace, hATP, Kraftwerk, mFAR are quadratic placers.

2. Partitioning based placers : Capo [18], Feng Shui [1] and Ntuplace [6] are recursive bipartitioning based placement approaches.

3. Pure simulated annealing based placers have become less popular in recent years, due to high run times. While annealing produces excellent results for small problems, it is used sparingly for local optimization.

4. Hybrid placers : Dragon [19] and Ntuplace2 [10] fall into this category. Dragon combines simulated annealing with recursive bipartitioning where as, NTUPlace2 combines analytic and bipartitioning based placement approaches.

Analytic placement tools have made significant advances recently, and the approach is widely used by industry groups. In particular, a patented method by Naylor[8] has received a great deal of attention. The approach by Naylor uses a *log-sum-exponent* function to represent wire length. In addition to the objective function, an analytic approach must also integrate constraints of some sort to avoid module overlap.

One of the best known implementations of [8] is APlace[11]; this tool has the best published results on the ISPD2005 benchmarks. It optimizes a non-linear function that combines the wire length and density objectives. It uses a smooth bell-shaped density potential function. The potential of a module being in its neighboring grids is high. As we move away from the module's current location, the potential fades smoothly. The potential of each module in its neighboring bins is computed. The potential for each grid is then transformed into a smooth density penalty function. Basically, regions with high overlap will be influenced by the potential of many modules and thus, will be penalized more. This penalty function is combined with the log-sum-exponent wire length function, stated in equation (1) later, and the overall objective is minimized. The objective function is smooth, continuously differentiable and hence easier to optimize. Iterative optimization of this objective leads to a placement that has relatively little overlap, allowing easy placement legalization. The nature of the penalty function slows convergence, however, resulting in extremely high run times.

In this paper, we present a new global placement approach for designs with large amounts of white space and fixed obstacles. Our placement tool, which we refer to as *Vaastu*, is fast and produces high quality results in terms of half perimeter wire length. Our key contribution is an iterative global placement algorithm resulting from the unification of continuous and discrete optimization techniques. Module spreading is accomplished through additional forces suggested by network flow algorithms, or a cut line shifting algorithm that is used in the last few iterations. The network flow algorithms incorporate the more realistic half-perimeter wire length based cost. Methods for speeding up the network flow algorithms are also discussed.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

The rest of the paper is organized as follows: In section II we briefly describe the analytic formulation used in our approach. Section III includes the details of our approach. Experimental results are presented in section IV. And finally, conclusions are drawn in section V.

## II. Analytic Formulation

We use the *log-sum-exponent* function to represent wire length in our analytic placer. This function is taken directly from the patent in [8]. APlace [11] and mPL [5] also use this function. To make this paper self-contained, we briefly describe it in this section. This function more accurately represents the half perimeter wire length (HPWL) as compared to the quadratic objective and therefore, was preferred in our approach.

Let, $\mathcal{N}$ denote the set of nets in the design, $\nu$ denote a net in $\mathcal{N}$, $m_i$ denote a module (cell/macro) in $\nu$ and $(x_i^\nu, y_i^\nu)$ denote the location of the pin of module $m_i$ corresponding to net $\nu$. The wire length for a placement $P$ as approximated by the *log-sum-exponent* function is given as follows:

$$wl(P) = \alpha \sum_{\nu \in \mathcal{N}} (log \sum_{m_i \in \nu} e^{(x_i^\nu/\alpha)} + log \sum_{m_i \in \nu} e^{(-x_i^\nu/\alpha)} + log \sum_{m_i \in \nu} e^{(y_i^\nu/\alpha)} + log \sum_{m_i \in \nu} e^{(-y_i^\nu/\alpha)}) \quad (1)$$

where, $\alpha$ is the smoothing parameter. Small values of $\alpha$ bring the objective function value closer to the half-perimeter wire length objective.

## III. Approach

Like most current academic placers, our placement approach is divided into 3 subcategories: $(i)$ Global Placement, $(ii)$ Legalization and $(iii)$ Detailed Placement. This paper addresses the global placement problem. The job of the global placer is to find a uniform distribution of modules across the placement area. There are numerous objectives, but in this paper, we will concentrate on minimization of half-perimeter wire length as the primary objective.

### A. Motivation

While analytic formulations can represent the wire length objectives of placement well, a traditional challenge has been in addressing overlap. A trivial solution to an unconstrained analytic formulation, in the absence of fixed obstacles, would be to simply have all modules overlap. In the presence of fixed obstacles, modules get attracted towards the obstacles, but the placement generated by an unconstrained optimizer still contains a nontrivial amount of overlap. Over the years, many methods of overlap removal have been explored. Spreading forces or fixed points are widely used. A goal of our work is to develop methods to encourage placement spreading without degrading wirelength or slowing convergence.

The main contribution of our work is to utilize discrete optimization techniques to expedite the process of module spreading. In our work, we have explored a network flow based approach to removing overlap from an analytic placement solution. Our motivation for using network flows is as follows.

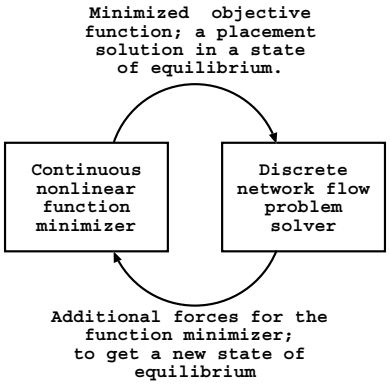


Fig. 1. Interaction between the two main optimizers in our global placement approach.

In designs with large amounts of white space, the solution space of the problem is increased. Biasing an analytic approach towards an overlap-free placement must be done carefully to avoid degrading wire length, and the underlying algorithm needs to be both robust and fast.

As with many analytic approaches, we add additional forces to spread the placement. We wish to only add forces that can guarantee that the direction in which the spreading is being done has enough space to hold the modules being spread, but at the same time, not affect the wire length. Network flow based techniques seemed to meet these objectives. The main advantage is that these algorithms can not only solve problems of supply-demand nature but also incorporate cost in the formulation. Indeed, a minimum cost flow algorithm that considers half-perimeter wire length as cost is one of the strong points of our approach.

### B. Outline

Our approach incorporates continuous and discrete optimization techniques, as sketched in figure 1. The two key elements are enumerated below:

1. A *continuous* nonlinear function minimizer, known as the Analytic Solver (AS), that minimizes equation (1). In our implementation, we use the L-BFGS-B solver[21].

2. A *discrete* Network Flow problem Solver (NFS).

The placement produced by the AS can be thought of as one in a state of equilibrium. In order to spread the modules, we need to disturb this equilibrium. This role is played by the NFS. Its job is to supply additional forces to the AS so that the equilibrium can be disturbed. The AS uses these additional forces and finds a placement with a new state of equilibrium.

Figure 2 shows the flow chart of the algorithm. The presence of a large number of movable modules makes it impractical to solve the network flow instance extracted from a flat view of the design. This necessitates the use of some clustering technique. Once clustering is done, the placement region is divided into uniform sized bins and a network flow problem instance is extracted and solved by the NFS. New forces are created for modules using the solution found by the NFS. This information

is supplied to the AS through a technique called *Anchoring*. We start with a small number of clusters and increase the number of clusters in each iteration *i.e.* decrease the cluster size in each iteration. After a few iterations, the graph size increases to a stage where it becomes impractical to run the NFS, as the number of clusters and bins is too large. Rather than incurring a high run time, we use a fast but effective technique, known as *cut line shifting* [13], for spreading the modules at the later stages of global placement.

### C. Physical Clustering

Placement tools such as APlace and mPL use hypergraph clustering techniques to reduce the problem size, where net-connectivity is considered to form clusters. Unlike these approaches, we use *only* the physical location of modules obtained from the AS to form clusters. Therefore, this step is called *Physical Clustering*. Note that clustering is used in our approach only to speed up the NFS. The AS is always run on the flat view of the design.

Let $N_{clust}$ be the desired number of clusters and $M$ be the set of all movable modules. Then, the desired area per cluster, say $dcarea$, is equal to $\frac{M_{area}}{N_{Clust}}$. Modules in $M$ are sorted by either their $x$ or $y$ coordinates depending on whether the bounding box of $M$ has greater width or height respectively. After sorting, a vertical (for higher width) or horizontal (for higher height) scan line is run across the placement and a split point is found where the modules on each side have almost same total area, but a multiple of $dcarea$. $M$ is split into 2 parts, say $M_1$ and $M_2$, at the split point. The process is then repeated on $M_1$ and $M_2$ until we end up with a set of modules with area equal to or less than $dcarea$. The set of modules at the bottom level of the above process gives us the physical clusters.

The size of clusters is kept as close to uniform as possible. By doing so, a one-to-one correspondence can be maintained between clusters and bins. The main advantage of doing this is that we can use efficient algorithms for solving the network flow problem instances.

### D. Network Flow Algorithms

Once the physical clusters are formed, the placement region is divided into equal sized bins. The NFS extracts a network flow problem instance from the placement and solves it. NFS consists of two main network flow algorithms *viz* minimum cost flow (MCF) and maximum flow (MF).

The minimum cost flow framework has been used in a variety of placement tools, for tasks such as legalization[3], detailed placement[7], or for partitioning space to remove overlap[4].

#### D.1 Model

We create a transformed bipartite graph $G = (V,E)$ between the clusters and bins as follows. Let $C$ and $B$ denote the set of clusters and bins.

- The node set $V = C \cup B \cup \{s,t\}$ where $s$ and $t$ are *supersource* and *supersink* nodes.
- The edge set $E = (C \times B) \cup (\{s\} \times C) \cup (B \times \{t\})$

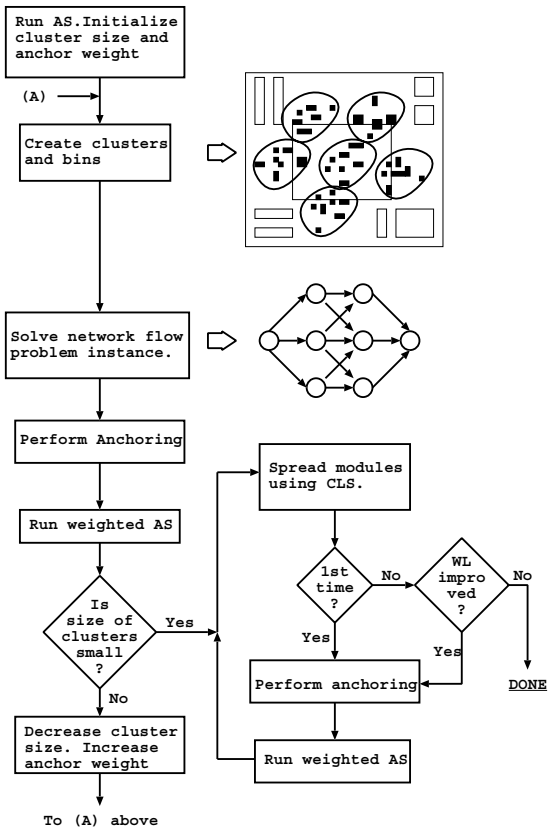


Fig. 2. Flow chart of the global placement approach.

- Each edge $e \in E$ has the following variables associated with it: $(i)$ $u_e$, the capacity or an upper bound on flow $(ii)$ $f_e$, the actual flow through this edge $(iii)$ $w_e$, the cost.
- For each edge $e \in (C \times B)$, $u_e = \infty$ and $w_e$ is the cost of assigning the corresponding cluster to the bin, which will be discussed in section D.2. For each edge $e \in (\{s\} \times C) \cup (B \times \{t\})$, $u_e = 1$ and $w_e = 0$.
- The node $s$ has a supply of $|C|$ and the node $t$ has a demand of $-|C|$.

We need to find the minimum cost assignment of clusters to bins. This problem is a special case of the minimum cost flow problem and is known as *weighted bipartite matching* or the *transportation problem*.

Various algorithms exist to solve this problem. We have used the *successive shortest path* (SSP) algorithm in our approach [2]. To make the algorithm efficient, we impose integrality constraints on all variables and a nonnegativity constraint on all the cost variables. SSP utilizes a shortest path algorithm internally. The nonnegativity constraint on cost variables lets us use an efficient implementation of Dijkstra's algorithm; the shortest path can be found in $O(|E| + |V|log|V|)$ time using a priority queue. Details of the SSP algorithm are not discussed here due to space limitations. The algorithm has a time complexity of $O(|C| \times (|E| + |V|log|V|))$.

### D.2 Assignment Cost

The cost $w_e$ of an edge $e = (c,b) \in (C \times B)$ is the cost of assigning the cluster $c = \{m_1, m_2, \ldots, m_n\}$ to bin $b$, where $m_1, m_2, \ldots, m_n$ are the modules in cluster $c$. We model this cost in terms of the gain (increase) in HPWL produced if each module in $c$ is relocated to the center of $b$ as opposed to its original location. Let $\mathcal{N}_c$ denote the set of all nets belonging to cluster $c$. Then the gain is computed as below:

$$gain(e) = gain(c,b) = \sum_{\nu \in \mathcal{N}_c} (HP(\nu) - HP^b(\nu)) \qquad (2)$$

where, $HP(\nu)$ is the HPWL of net $\nu$ considering the initial location of modules in $\nu$ and $HP^b(\nu)$ is the HPWL of $\nu$ with respect to bin $b$ and is calculated as follows: if a module $m_i$ is such that $m_i \in c$ and $m_i \in \nu$, then the location of $m_i$ is the center of the bin $b$, otherwise the location of $m_i$ is its initial location.

We wish to maximize the gain stated in equation (2) for the entire assignment. If the cost of assigning cluster $c$ to bin $b$ is expressed as in equation (3), then minimizing the cost would be the same as maximizing the gain. Thus, a MCF solution for a graph with these costs suits our needs.

$$w(e) = w(c,b) = -gain(c,b) \qquad (3)$$

We mentioned in section D.1, that the costs of all edges should be nonnegative. To respect this constraint, we calculate the least cost of all edges in $(C \times B)$, say $MIN(w(e))$. The cost of all edges in $(C \times B)$ are then normalized as follows:

$$w(e) = w(e) - MIN(w(e)), \forall e \in (C \times B) \qquad (4)$$

### D.3 Sliding window based improvement

After running MCF on the entire problem, we perform window based optimization where we select a small number of bins and readjust the clusters assigned to them by running MCF on the smaller graph. The window is slid across the entire placement region in this step. The modeling and cost function is the same in this step as in the global problem.

The cost function described in the previous section bears some degree of inaccuracy. Consider two clusters $c_A$ and $c_B$ whose modules share common nets. While finding the assignment cost for $c_A$, $c_B$ is assumed to be stationary and vice versa. However, after the assignment is done, both move to new locations. Thus, our prior assumption was wrong. The degree of inaccuracy is very high in the initial iterations when the degree of overlap is very high, as clusters encounter large displacements. As modules are spread, the inaccuracy is reduced.

Since a good assignment is crucial to the success of our placement algorithm, window based optimization plays a greater role in our approach as opposed to when used during detailed placement. It gives us huge improvements, especially during the initial iterations. Since all the clusters outside the window are fixed, the cost function has a higher degree of accuracy as compared to the case where every cluster is mobile.

### D.4 Speed Up : Radius

The number of edges in our graph ($|E|$) is $(|C|+|B|+|C| \times |B|)$. Since $(|B| \geq |C|)$ is a necessary condition for a feasible assignment, $|E|$ is larger than $|C|^2$, which can be roughly thought of as a function of $|V|^2$. If every cluster is modeled against every bin, the MCF algorithm could be extremely time consuming, and impractical after only a few iterations. The complexity of Dijkstra's shortest path algorithm, in this case, would be $O(|V|^2)$.

We therefore propose the idea of *radius*, which is a small predetermined constant. For example, if $radius = k$, then we only try assigning a cluster $c$ to the closest $k$ bins. The distance is computed from the center of mass of modules in $c$ to the center of each bin. This approach, however, has a drawback that no feasible flow might be present in the graph. Flow feasibility can be checked by running a maximum flow algorithm (which can be solved much more efficiently than MCF) on the graph prior to running the MCF algorithm. As the modules are spread, there is less dispute between clusters for access to the same bins. So radius can be reduced in the final iterations. If we hit a situation where no feasible flow exists, we increase the radius and start the iteration afresh.

Bins outside of the radius of any cluster are not considered by the minimum cost flow algorithm. Due to overlap, the closest bin sets for many clusters intersect. As a result, the number of bins in the graph ($|B'|$) is smaller than $|B|$. With a radius of $k$, $|E|$ is equal to $(|C| + |B'| + |C| \times k)$. This results in a considerable improvement in the run time of the shortest path and hence the MCF algorithm.

Note that the idea of radius is only used while solving the global flow problem. The window based optimization is always solved by mapping all clusters with all bins.

### D.5 Speed Up : Maximum Flow

In the last few iterations, the run time for NFS starts increasing, even with the idea of radius. To speed-up the placement algorithm, we switch to using maximum flow (MF) instead of MCF, to find an assignment.

Our graph comes under the special case of *unit capacity bipartite networks*. The MF problem can be solved very efficiently for this special case.

Note that this assignment will no longer be a minimum cost assignment. However, since each cluster is mapped with the closest bins, this assignment is not completely absurd. Moreover, we can still run window based optimization considering assignment costs to improve upon the solution found by MF. Experiments show that this results in only a small loss in quality but considerable improvements in run times, especially for the bigger designs.

## *E. Anchoring*

The result of NFS is passed to AS using a technique we refer to as *anchoring*; this is similar in spirit to the fixed points used in tools such as mFar[9]. Once we find the cluster assignment using MCF/MF, we create an *anchor* for each module at the center of the bin to which its parent cluster is assigned. A *pseudonet* is then created between each module and its anchor. The job of anchors and pseudonets is to simply create additional constraints for the AS to facilitate spreading.

Pseudonets are 2-pin nets and carry less weight than the original nets in the design. The reason for this is obvious; we do not want to encourage the AS to optimize the pseudonets over the original nets. Initially, the pseudonet weight is kept very

low. Due to very high module overlap, the assignment found by NFS is not very good. As the modules spread, NFS accuracy increases and it is able to find good assignments. Since module spreading is an objective as important as the wire length objective for us, we increase the weight of pseudonets, linearly, in successive iterations, but always keep it less than the weight of the original nets (which is 1 for all nets).

After anchoring, we have a different problem with additional constraints, created by the pseudonets, for AS. The solver is called again and a minima for the objective function is found which gives us a new placement. NFS is called again until the cluster size is very small (a few modules per cluster).

Note that in every iteration, the old anchors/pseudonets are thrown away and fresh anchors are created. Since pseudonet weight strictly increases in each iteration, the placement can not collapse.

### F. Parameters

In this section, we summarize the parameters in our global placer and their variation with iterations. These values are determined experimentally. The placer is not overly sensitive to these values. Some modification in these values does not degrade the solution quality significantly.

1. *Number of clusters*: The desired cluster area is initialized to 2000 times the average movable module area and decreased uniformly to 25 times the average movable module area in the last iteration. Initially with bigger clusters, modules are assigned to bigger bins. As we get smaller clusters, modules slowly get an opportunity to seep into the narrow regions in between obstacles.

2. *Radius*: Radius is kept at 100 bins per cluster in the initial iterations and reduced to 50 bins per cluster during the last few iterations.

3. *Pseudonet weight*: Pseudonet weight is increased uniformly from 0.1 to 0.5. The weight of original nets is kept constant at 1.0 throughout. Variation in this range produces slightly different results. However, increasing the upper limit degrades the solution quality, as expected. Reducing it does not spread the modules quickly.

4. *Alpha*: The smoothing parameter $\alpha$ for the AS is kept proportional to the bin dimensions.

### G. Cut line shifting

NFS algorithms run on clusters and give us anchors at the center of the bins. The advantage of NFS algorithms is that HPWL-based cost can be incorporated into the model. At the lowest placement levels, however, other methods produce comparable results but have lower computational complexity.

At the bottom level, we use another technique, called *cut line shifting (CLS)* to aid us in finding anchors. This was first proposed in [13], and was used for white space distribution to improve routability. Later on, it was used in [12] for incremental cell spreading after buffer insertion.

We use it in our approach for module spreading during the last stage of our global placement algorithm. This technique is completely geometric in nature and considers only module locations. But this is not a concern to us. Most of the spreading has been done using the NFS and we just need to have some fine-grained control in the last stage. This technique, being geometric in nature, is extremely fast, which is just what we want towards the termination of our algorithm.

We briefly describe the algorithm here. A slicing tree is first created by recursively partitioning the modules into 2 halves by inserting vertical and horizontal cut lines, thus creating separate regions. Vertical/horizontal cut lines are inserted if the regions at a level of the slicing tree are wide/tall respectively. Another point to note here is that the partitioning is done solely on the basis of module locations. Regions are partitioned until they contain a small number of modules. Once this is done, the area overflow in each region is measured. A region has overflow if the total module area is greater than the free space in the region and underflow vice versa. The overflow/ underflow is then propagated from the bottom level regions of the slicing tree to the top level regions. We then start at the top 2 regions and distrbute the available white space proportionally to each region depending on its need. The white space is distributed from the top level to the bottom level of the tree. Each time, the cut line between regions is shifted left/right for a vertical cut or down/up for a horizontal cut to adjust for the white space allocated to the regions.

CLS increases the wire length of the placement; however there is still some scope for wire length optimization. Until now, we were spreading modules using center of bins as anchor locations. Thus, modules belonging to a cluster had the same anchor locations. After spreading the modules using the CLS algorithm, we again create anchors, but this time, at the location suggested by CLS. Thus each module gets a different anchor location. This location is more accurate for individual modules, as compared to having a single anchor location for a bunch of modules.

Due to increased flexibility in moving modules, wire length after calling the AS decreases for a few iterations, before becoming stable. We therefore run a few iterations of AS and CLS till be get a significant improvement in HPWL. Finally, we call the CLS algorithm for one last time and terminate the global placement. At this time, modules are spread well enough and we are ready to call the legalizer and detailed placer.

### H. Legalization and Detailed Placement

After the global placement, we use the legalizer and detailed placer published in [17] to gauge the amount of wire length improvement we can get for legal placements and comparing our work with other placers.

## IV. Experimental Results

We performed our experiments on the ISPD2005 placement contest benchmarks [14]. Table IV shows the benchmark characteristics. In all benchmarks, except *bigblue3*, macros are fixed and standard cells are movable. *Bigblue3* has 2485 movable macros. The utilization ranges between 27% and 57% approximately. All our experiments were performed on an AMD

Athlon Dual Core machine with 2.2/1 GHz processors. We used the optimization software L-BFGS-B [21] in our analytic solver to minimize the log-sum-exponent objective function.

| Benchmark | #movable | #fixed | #nets | Util.(%) |
|---|---|---|---|---|
| adaptec1 | 211k | 543 | 221k | 57.34 |
| adaptec2 | 254k | 566 | 266k | 44.32 |
| adaptec3 | 451k | 723 | 467k | 33.52 |
| adaptec4 | 495k | 1329 | 516k | 27.14 |
| bigblue1 | 278k | 560 | 284k | 44.67 |
| bigblue2 | 535k | 23084 | 577k | 37.78 |
| bigblue3 | 1096k | 1293 | 1123k | 56.48 |
| bigblue4 | 2169k | 8170 | 2230k | 44.29 |

TABLE I
BENCHMARK CHARACTERISTICS

| BM | HPWL ($\times e6$) | | | | Run time (sec) | |
|---|---|---|---|---|---|---|
| | GP (I) | LDP (I) | GP (II) | LDP (II) | Total (I) | Total (II) |
| a1 | 84.11 | 81.61 | 85.73 | 81.53 | 1272 | 920 |
| a2 | 93.19 | 89.92 | 96.70 | 92.97 | 1665 | 1204 |
| a3 | 231.58 | 223.87 | 230.93 | 219.39 | 4359 | 2676 |
| a4 | 195.42 | 190.15 | 199.48 | 192.06 | 4751 | 2330 |
| b1 | 102.81 | 99.17 | 100.09 | 98.09 | 2089 | 1406 |
| b2 | 157.11 | 151.22 | 156.79 | 153.43 | 5317 | 3288 |
| b3 | 389.11 | 371.54 | 385.87 | 370.72 | 14295 | 6434 |
| b4 | - | - | 868.28 | 828.25 | - | 17011 |
| Avg. | 1.00 | 1.00 | 1.005 | 1.003 | 1.68 | 1.00 |

TABLE II
HPWL AND RUN TIME COMPARISON FOR 2 GLOBAL PLACEMENT (GP) FLOWS; (I) (MCF), (II) (MCF + MF). COLUMNS 3/5 SHOW HPWL AFTER RUNNING LEGALIZATION/DETAILED PLACEMENT (LDP) ON THE PLACEMENTS IN COLUMNS 2/4. COLUMNS 6/7 SHOW THE TOTAL RUN TIMES FOR (GP + LDP) IN FLOWS (I)/(II). AVERAGES IN LAST ROW ARE COMPARED BETWEEN COLUMNS 2/4, 3/5 AND 6/7.

We tried two different flows for our *Vaastu* global placer. In flow (I), we used Minimum Cost Flow (MCF) until the bottom level (where physical cluster size is roughly 50 times the average area of a standard cell). In flow (II), we used Maximum Flow (MF) in the last few iterations instead of MCF. In this flow, our bottom level physical cluster area was down to 25 times the average standard cell area. For flow (I), the numbers for the design *bigblue4* are not reported, as we did not have access to a machine with enough memory to complete the design.

### A. Wire length and run time break-ups

Table A shows the HPWL and run time break-ups. HPWL is reported for the two flows after global placement (GP) and after legalization and detailed placement (LDP). The run time columns show the total run times (GP + LDP) for the two flows. Placement run times scale almost linearly with the problem size.

The loss in HPWL in flow (II) running (MCF + MF) is very small. However, the savings in run time are large. This shows the effectiveness of using MF and the sliding window MCF technique.

The detailed placer is able to get about $3-4\%$ improvements over the global placement wire length in both cases. This is a fair enough gain and shows that cells are spread well enough in terms of both density and quality leaving enough scope for the detailed placer to do a fast overlap removal and local optimization.

### B. Comparison with other placers

Table B shows the comparison of HPWL and run times of our placer with other state-of-the-art placers. mPL6 results are taken from [5]. All other results are taken from [11]. APlace and Capo were run on a 1.6GHz machine as reported in [11]. So these run times are not directly comparable.

Our placer's flow (II) needs less than 10 hours of run time for all the benchmarks. On average, wire length is only 3.7% larger than the best published results on these benchmarks.

## V. CONCLUSION

A fast and effective global placement approach for large designs with fixed obstacles and abundant white space is presented. The approach incorporates a log-sum-exponent function based analytic solver. Module spreading is done iteratively by treating the problem as discrete and then generating additional spreading forces for the analytic solver. Network Flow algorithms, which incorporate the more realistic half-perimeter wire length objective, are used to do this. Techniques for speeding-up the network flow algorithms are suggested for fast convergence. Our placer produces competitive results in terms of wire length compared to state of the art academic placers, while running much faster. We are currently integrating the new *Vaastu* global placement approach with our existing *feng shui* physical design tools. Binary versions of the tools will be available through our research group web site.

**Acknowledgements:** We would like to thank a number of people for their assistance in this work. Prof. Cheng-Kok Koh of Purdue and Dr. Chen Li of Magma provided many helpful comments and suggestions, as well as source code to some of their tools. Prof. Chris C.-N. Chu provided FastDP for legalization and detailed placement. We use a port of the L-BFGS-B[21] analytic solver, originally developed by Ciyou Zhu, R. H. Byrd, P. Lu-Chen, and J. Nocedal, from Argonne National Laboratory and Northwestern University. The solver port was done by Taku Kudo from Nara Institute of Science and Technology. We would also like to thank Dr. Gi-Joon Nam of IBM for organizing the ISPD placement contests, and the research groups who have participated in them.

## REFERENCES

[1] A. R. Agnihotri, S. Ono, C. Li, M. C. Yildiz, A. Khatkhate, C.-K. Koh, and P. H. Madden. Mixed block placement via fractional cut recursive bisection. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):748–761, 2005.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

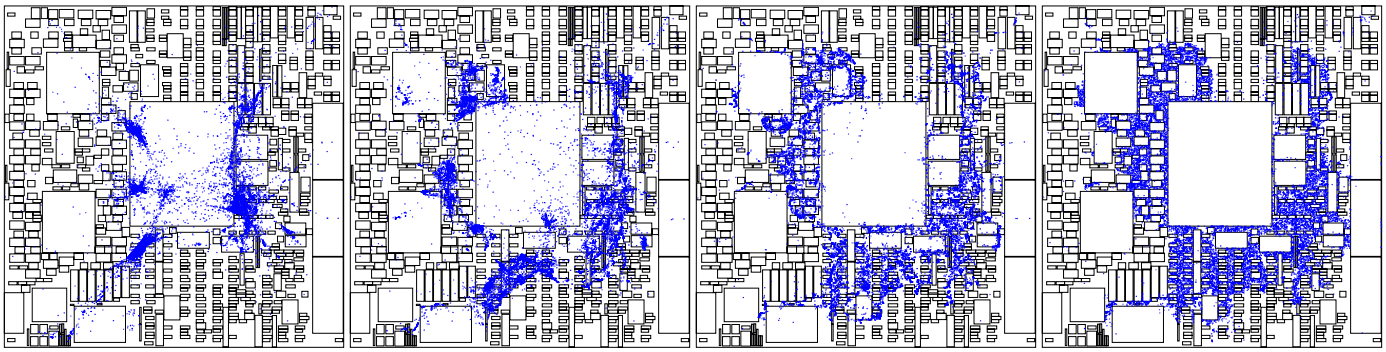

Fig. 3. Placement snapshots during spreading on the design Adaptec3

| Placer | HPWL($\times e6$) on benchmark | | | | | | | | Avg. | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue2 | bigblue3 | bigblue4 | HPWL | CPU |
| Vaastu(I) + FastDP[17] | 81.61 | 89.92 | 223.87 | 190.15 | 99.17 | 151.22 | 371.54 | - | 1.036 | 9.4h |
| Vaastu(II) + FastDP[17] | 81.53 | 92.97 | 219.39 | 192.06 | 98.09 | 153.43 | 370.72 | 828.25 | 1.037 | 9.8h |
| APlace[11] | - | 87.31 | - | 187.65 | 94.64 | 143.82 | 357.89 | 833.21 | 1.00 | 113.2h |
| mPL6[5] | 78.7 | 96.5 | 221 | 200 | 98.7 | 154 | 354 | 846 | 1.048 | 30.18h |
| mFAR[9] | - | 91.53 | - | 190.84 | 97.70 | 168.70 | 379.95 | 876.28 | 1.06 | - |
| Dragon[19] | - | 94.72 | - | 200.88 | 102.39 | 159.71 | 380.45 | 903.96 | 1.08 | - |
| FastPlace[20] | - | 107.86 | - | 204.48 | 101.56 | 169.89 | 458.49 | 889.87 | 1.16 | - |
| Capo[18] | - | 99.71 | - | 211.25 | 108.21 | 172.30 | 382.63 | 1098.76 | 1.17 | 37.8h |
| Ntuplace[6] | - | 100.31 | - | 206.45 | 106.54 | 190.66 | 411.81 | 1154.15 | 1.21 | - |
| Feng Shui[1] | - | 122.99 | - | 337.22 | 114.57 | 285.43 | 471.15 | 1040.05 | 1.50 | - |
| Kraftwerk[16] | - | 157.65 | - | 352.01 | 149.44 | 322.22 | 656.19 | 1403.79 | 1.84 | - |

TABLE III

HALF PERIMETER WIRE LENGTH (HPWL) COMPARISONS WITH OTHER PLACEMENT TOOLS. RUN TIMES AND BINARY VERSIONS OF MANY OF THE TOOLS ARE NOT AVAILABLE. TOTAL CPU FOR OUR PLACER'S FLOW (I) IS REPORTED FOR THE FIRST 7 BENCHMARKS WHERE AS FOR FLOW (II) IT IS REPORTED FOR ALL THE BENCHMARKS. MPL6 RUN TIMES ARE REPORTED FOR ALL BENCHMARKS


[3] U. Brenner, A. Pauli, and J. Vygen. Almost optimum placement legalization by minimum cost flow and dynamic programming. In *Proc. Int. Symp. on Physical Design*, pages 2–9, 2004.

[4] U. Brenner and M. Struzyna. Faster and better global placement by a new transprotation algorithm. In *Proc. Design Automation Conf*, pages 591–596, 2005.

[5] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. mpl6: enhanced multilevel mixed-size placement. In *Proc. Int. Symp. on Physical Design*, pages 212–214, 2006.

[6] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang. Ntuplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proc. Int. Symp. on Physical Design*, pages 236–238, 2005.

[7] K. Doll, F. M. Johannes, and K. J. Antreich. Iterative placement improvement by network flow methods. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(10):1189–1200, 1994.

[8] W. Naylor et al. US patent 6,301,693: Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer, 2001.

[9] B. Hu and M. Marek-Sadowska. Multilevel fixed-point-addition-based vlsi placemen. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(8):1188–1203, August 2005.

[10] Z.-W. Jiang, T.-C. Chen, T.-C. Hsu, H.-C. Chenz, and Y.-W. Chang. Ntuplace2: A hybrid placer using partitioning and analytical techniques. In *Proc. Int. Symp. on Physical Design*, pages 215–217, 2006.

[11] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proc. Int. Conf. on Computer Aided Design*, pages 891–898, 2005.

[12] C. Li, C.-K. Koh, and P. H. Madden. Floorplan management: Incremental placement for gate sizing and buffer insertion. In *Proc. Asia South Pacific Design Automation Conf.*, pages 349–354, 2005.

[13] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden. Routability-driven placement and white space allocation. In *Proc. Int. Conf. on Computer Aided Design*, pages 394–401, 2004.

[14] G.-J. Nam, C. J. Alpert, P. G. Villarubbia, B. Winter, and M. C. Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proc. Int. Symp. on Physical Design*, pages 216–220, 2005.

[15] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarubbia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(4):678–691, 2006.

[16] B. Obermeier, H. Ranke, and F. M. Johannes. Kraftwerk: a versatile placement approach. In *Proc. Int. Symp. on Physical Design*, pages 242–244, 2005.

[17] M. Pan, N. Viswanathan, and C. C.-N. Chu. An efficient and effective detailed placement algorithm. In *Proc. Int. Conf. on Computer Aided Design*, pages 48–55, 2005.

[18] J. Roy, D. Papa, A. Ng, and I. Markov. Satisfying whitespace requirements in top-down placement. In *Proc. Int. Symp. on Physical Design*, pages 206–208, 2006.

[19] T. Taghavi, X. Yang, B-K Choi, M. Wang, and M. Sarrafzadeh. Dragon2006: blockage-aware congestion-controlling mixed-size placer. In *Proc. Int. Symp. on Physical Design*, pages 209–211, 2006.

[20] N. Viswanathan and C. C.-N. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):722–733, May 2005.

[21] C. Zhu, R. H. Byrd, and J. Nocedal. L-BFGS-B, FORTRAN routiens for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.