Hierarchical Placement and Floorplanning in BEAR

WAYNE WEI-MING DAI, MEMBER, IEEE, BERNHARD ESCHERMANN, ERNEST S. KUH, FELLOW, IEEE, AND MASSOUD PEDRAM

Abstract—**In this paper hierarchical placement and floorplanning algorithms for rectangular blocks are described. It is implemented as part of the BEAR building block layout system developed at the University of California at Berkeley. The algorithm combines the goal orientation of top-down approaches with the block orientation of bottom-up techniques. The result is a "meet in the middle" strategy. It considers the mutual dependency between placement and routing explicitly by incorporating a novel method of hierarchical routing area estimation. If the layout includes flexible blocks, the placement result can be further optimized by resizing these blocks subject to constraints on their areas and aspect ratios. Placement and floorplanning are refined more and more (with possible topological change) as routing proceeds. Global routing is updated incrementally to eliminate the need for iterations between placement and routing, thus achieving a more uniform design flow.**

I. Introduction

INTEGRATED circuit fabrication technology has advanced rapidly over the last three decades. Three major design styles—gate arrays, standard cells, and general cells—targeted toward different applications have evolved. Hierarchical decomposition is necessary in any of these approaches to deal with the complexity of VLSI circuits. At some level of abstraction the different parts of a circuit, be they developed manually, with module generators or composed of standard cells, can be viewed as rectangular or rectilinear objects with given shapes and I/O interfaces. These objects then have to be placed on a plane, oriented in one of the eight possible ways and connected with each other at fixed terminal locations.

The objective of the placement is to provide an arrangement of blocks that—after having been routed—fits into an enclosing rectangle of minimum area with given height, width, or aspect ratio. To get a high performance circuit, a concurrent goal is needed to minimize the interconnection length. In floorplanning, shapes of some of the blocks can be varied to a certain extent in order to reduce the layout area or the wire length.

Over the last few years many programs have been developed to solve these problems. (See [22] for an overview.) Placement and floorplanning are often solved as separate problems. In addition, placement and routing are done independently. Because of the interdependency of these steps, many iterations are necessary in order to obtain a satisfactory layout solution.

In the BEAR layout system, we perform the placement, routing area allocation, and block resizing in one step. As a result, high quality floorplanning solutions are obtained without need for iterations. After global routing, this floorplanning result may be further optimized by a global spacing and/or global shape optimization.

Manuscript received March 16, 1989. This work was supported in part by the Semicomputer Research Corporation and by the National Science Foundation under Grant MIP-85-06901. This paper was recommended by Associate Editor A. Dunlop.

W. W.-M. Dai is with the Board of Studies of Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064.

B. Eschermann is with the Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Germany.

E. S. Kuh and M. Pedram are with the Department of Electrical and Computer Science, University of California at Berkeley, Berkeley, CA 94720.

IEEE Log Number 8930755.

II. Overview

BEAR is a second generation macrocell-based layout system being developed at the University of California, Berkeley. The system takes advantage of our experience with BBL (Berkeley building-block layout system [3]) and feedback from the industry. Our goal is to provide automatic and interactive features to lay out a chip in both top-down and bottom-up physical design environments.

As a first step, we generate a hierarchical representation of the problem. Blocks that are strongly connected with each other are clustered together in clusters of some maximum size. To avoid a block shape mismatch in the cluster that makes it difficult to find a good placement for blocks in that cluster, the shapes of the blocks are also considered. This step is recursively repeated and produces the different hierarchical levels of a *clustering tree* (see [6], [10].)

In the placement step, this tree is traversed *top − down* (see [6]). Given an overall shape goal and locations of the I/O terminals (I/O goals), at each level of the hierarchy all *topological possibilities* for the clusters on that level of the hierarchy are examined. A topological possibility refers to the assignment of clusters to *rooms* of a particular *floorplan template* (see [6]). At the leaves of the clustering tree, block orientations must be specified as well.

The objective function for choosing a particular possibility is a linear combination of geometry cost and connection cost. The geometry cost has two components: the area cost which accounts for the template area and the shape cost which accounts for cluster to room shapes that mismatch. The connection cost penalizes connections between nonadjacent clusters. The user controls the tradeoff between geometry and connection costs. The chosen topological possibility, in turn, sets the shape goals and the I/O goals for the next lower hierarchical level. The

same decision process is repeated till all leaf level blocks are placed (see Fig. 1).

This top-down strategy works well for floorplanning with perfectly flexible (or "soft") blocks whose shapes can be adjusted to the shapes of the rooms on the lowest level. However, it can lead to unfavorable results in the case of rigid (or "hard") blocks with fixed geometries since the objective function on higher hierarchical levels has very little information about the actual block shapes at its disposal. Decisions are not well founded at this stage. In this respect it resembles other top-down techniques like the min-cut algorithm presented in [16] which, in some sense, is a special case of our algorithm for $k = 2$.

To provide the geometry cost function on higher levels with more information, the clustering also passes shape information from the leaves toward the root of the tree. Additionally, during the top-down traversal of the tree, a lookahead is possible so that the decision on some level is not restricted by the available block shape information on the immediately following level of the hierarchy (see Section III).

For each of the topological possibilities the routing space necessary to implement the connections between various clusters is estimated. This area and its location together with the cluster areas and shapes are used to compute the value of the geometry cost function. It is also used to determine the goal shape for the clusters on the next lower level of the hierarchy. The routing area estimation (and allocation) is hierarchical, i.e., space for global connections is provided on higher levels of the hierarchy when the rough positions of clusters are known. The allocation of space for local connections is deferred until later in the process when the detailed block positions on the clusters are to be generated (see Section IV).

Placement defines *capacities* of the routing areas around the blocks. Global routing defines *densities* (net assignments) of the routing areas. After placement and global routing, we can change the density by rerouting or we can change the capacity by global spacing (compaction or decompaction). In order to achieve a high density of the final layout, we iterate these two operations to obtain a satisfactory match of capacity and density of the routing area before detailed routing. During global spacing, global routing is updated incrementally. A dynamic data representation [7], which unifies topological and geometrical information, is used to achieve an efficient implementation of such difficult operations.

In contrast to the constraint-graph approach, the *ridge spacing method* for global spacing [8] is composed of small steps which iteratively partition the layout into two parts and performs contracting or expanding only on the space between the two partitioned pieces. This approach is particularly desirable for global spacing because we would like to preserve the topology of the placement as much as possible. (The job of global spacing is to match a reasonably good placement and a reasonably good global routing.) If there were a significant mismatch between placement and global routing, we would have to redo at least one of them.

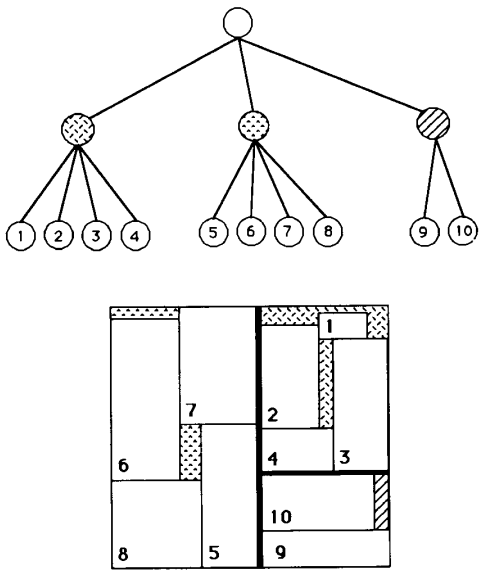
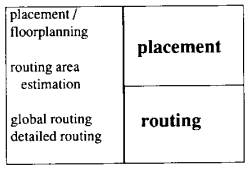


Fig. 1. This figure shows the clustering tree and the corresponding block packing (placement without routing area allocation). Leaves of the clustering tree represent building blocks.

"Horizontal" Layout Methodology



"Diagonal" Layout Methodology

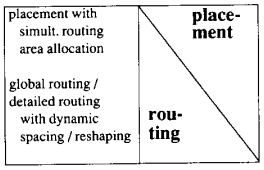


Fig. 2. BEAR has adopted the "diagonal" layout style. It considers the mutual dependency between placement and routing explicitly by simultaneous routing area estimation during placement and incremental global spacing and shape optimization during routing.

We can optimize shapes of the flexible blocks after global routing in order to minimize the layout area. The shape optimization is an improvement procedure that iteratively selects and resizes blocks. Since at this point the global routing information is known, routing areas can be accurately estimated. The shape optimizer uses these estimates to compute the longest paths through the layout surface and to determine the "best" block candidate for

resizing. The global routing information is incrementally updated from one iteration to the next (see Section V).

These considerations result in a layout methodology that deviates from the standard methodologies in some important aspects. Looking at Fig. 2, the term "diagonal methodology" as opposed to the conventional "horizontal methodology" seems appropriate. This approach also helps to reduce the *sensitivity* of the BEAR system to changes in the input data (see Section VI).

III. Placement: A "Meet in the Middle" Strategy

3.1. Representation of Connections

In order to evaluate various topological possibilities efficiently, the connectivity information has to be represented in a more versatile form than as n-point nets between geometrical pin locations. The accuracy required to obtain satisfactory results varies on different levels of the hierarchy.

Independent of the hierarchical level, nets with n terminals are represented by $n(n-1)/2$ connections between every pair of terminals. A spanning tree for this n-clique has $n-1$ edges. Every edge of the n-clique has a probability of $2/n$ to be in the spanning tree. Therefore, each of the edges is assigned a weight of $2/n$.

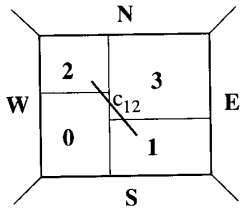
On the non-leaf level, all connections within the clusters are invisible. The connections between two clusters are measured from center point of one to center point of the other. (See Section III-3.) In this way all connections of current interest can be summed up in one matrix attached to the node being placed. The element c_{ij} in row i and column j of this *connectivity matrix* contains the sum of the weights of connections between clusters i and j (Fig. 3). If a cluster i has a link to another cluster outside the parent node, this link is split up into two links of pertinent weights in the applicable I/O-directions as shown in Fig. 4.

On the leaf level this representation is not exact enough since the optimal orientation of a block depends on its actual pin positions. (Each leaf node represents an actual building block.) However, it would be very costly to search through a list of all pin positions each time a new block orientation is evaluated. Some accuracy has to be given up to gain efficiency. A convenient scheme is to distinguish only between different pin directions. All the connections summarized in one number c_{ij} in previous levels of the hierarchy are now split up into a set of four numbers representing the connections between block j and each of the sides of block i. In this way the data structure does not have to be augmented; the information can be held by the leaf level connectivity matrices (see Fig. 5).

3.2. Target Shapes and Lookahead

In many cases the original heuristics [6] for computing the objective function on non-leaf levels do not lead to the best solution. More information about the "desirable" shapes must be available on higher levels of the hierarchy in order to increase the accuracy and the discriminating power of the placement objective function.[1] Some algorithms solve the problem by deriving the placement *bottom − up* [19], [15]: all possible combinations of block sizes are propagated up to the root of the tree where the combination with optimum aspect ratio is chosen. For slicing trees this approach is efficient but it restricts the set of obtainable solutions considerably.

Our approach is to compute the optimal shape goal for the lowest level clusters and to propagate this information



	1	2	3	N	W	S	E
0	c_{01}	c_{02}	c_{03}	c_{0N}	c_{0W}	c_{0S}	c_{0E}
1		c_{12}	c_{13}	c_{1N}	c_{1W}	c_{1S}	c_{1E}
2			c_{23}	c_{2N}	c_{2W}	c_{2S}	c_{2E}
3				c_{3N}	c_{3W}	c_{3S}	c_{3E}

Fig. 3. A particular topological possibility and its corresponding connectivity matrix are shown above. The element c_{ij} in row i and column j of this connectivity matrix contains the sum of the weights of connections between cluster i and cluster j (or I/O j).

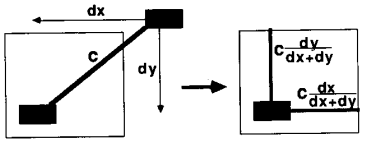


Fig. 4. If a cluster i has a link to another cluster outside the parent node, this link is split up into two links of pertinent weights as shown above.

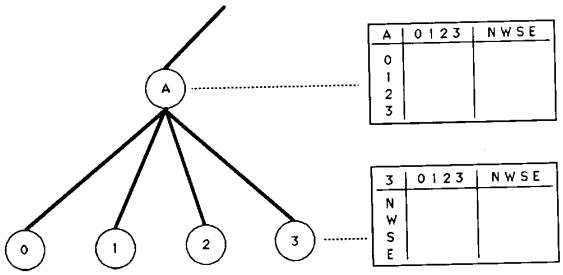


Fig. 5. Connectivity matrices attached to the nodes of the clustering tree are shown. The connectivity matrix attached to a leaf node is different from that attached to higher level nodes. All the connections summarized in one number c_{ij} in higher levels of the hierarchy are now split up into a set of 4 numbers representing the connections between block j and each of the sides of block i.

[1] An objective function is a good discriminator if it provides accurate early warning signals for all off-track nodes encountered along the path search toward an optimal solution.

recursively up the clustering tree. The optimal shape goal (*target shape*) of a cluster is derived by enumerating all topological possibilities to combine the cluster elements. During the top-down placement process, the geometry cost for a particular topological possibility is computed by treating the target shape for the corresponding node of the tree as its actual shape. This leads to more accurate and reliable geometry cost computation.

Some modifications to this scheme are necessary to make it work. Target shapes with aspect ratios far from 1 are useless even if they represent the optimal way to place a number of very long blocks with the same width. Undesirable aspect ratios have to be penalized. The objective function has to account for this effect. The more dead space a target shape contains, the lower its impact should be on the choice of a topological possibility. Real-world macrocell layout examples by and large consist of less than 50 blocks. Predictions for future "superchip" block counts are in the range of 100. In any case, with 5 blocks per cluster, 125 blocks can be accommodated in a clustering tree with a depth of 3. Let us denote the 4 levels of the clustering tree built for such a "superchip" as 1 (root level), 2, 3, and 4 (building block level). The target shapes on level 3 are optimal since the building block dimensions are known and all combinations of these blocks are evaluated in order to choose the best topological possibility. Target shapes only become questionable on level 2 because the optimal combination of optimal level 3 shapes need no longer be optimal. Therefore, if the clustering tree is of depth 3 (4 levels) we can avoid using the possibly suboptimal level 2 shapes by doing a 1-level lookahead and using the target shapes of level 3 (see Fig. 6).

Although the search tree is a shallow, its branching factor turns out to be very high. Under the assumption that the clustering tree is a tree in which every internal node has k children, an l-level lookahead has a complexity of [10][2]

$$O\left(\frac{n-1}{k-1} f(k)^{l+1} k^l\right).$$

Although this is still linear in the number of blocks n, $f(k)$ is very large (Table I). $f(k)$ is larger in the presence of target shapes and for nodes just above the leaf level.

[2]For a node in a k-tree, $f(k)$ topological possibilities have to be enumerated (lookahead $l = 0$). For each of the possibilities, each cluster element itself has $f(k)$ possibilities to be placed. Therefore, $kf(k)$ possibilities have to be examined. If the effect of the leaf level, where the lookahead has to stop, is neglected, then for a lookahead depth of l levels $f(k)(kf(k))^l$ evaluations of the objective function are necessary. A k-tree of depth d contains $n = k^d$ leaf nodes and:

$$1 + k + \cdots + k^{d-1} = \frac{k^d - 1}{k - 1}$$

non-leaf nodes (clusters). Hence, the complexity is bounded by

$$\frac{k^d - 1}{k - 1} f(k)(kf(k))^l = \frac{n-1}{k-1} f(k)^{l+1} k^l$$

which is the desired result.

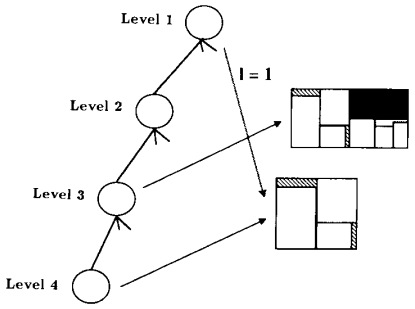


Fig. 6. If the clustering tree is of depth 3 (4 levels) we can avoid using the possibly suboptimal level 2 shapes by doing a 1-level lookahead and using the target shapes of level 3.

TABLE I
NUMBER OF TOPOLOGICAL POSSIBILITIES OF $f(k)$ FOR NON-LEAF CLUSTERS. k IS THE NUMBER OF SUBCLUSTERS

k	1	2	3	4	5
$f(k) = k! \times t(k)$	1	4	36	528	11040

This is because nodes with target shapes and leaf nodes have exact shapes and their various orientations must be considered in the optimization process. $t(k)$ in Table I is the number of different placement topologies, i.e., the number of placement templates with k rooms.

Even after restricting the clusters to sizes $k \leq 5$, it is necessary to find a scheme to speed up the program. Pruning of the search tree to reduce the effective branching factor is done with the help of the previous objective function. Since only solutions within some percentage range of the "optimum" are kept as candidates for further explorations, the search space is considerably reduced in size. (This "optimum" cost is initially computed by enumerating all topological possibilities with no lookahead.) Possibilities that are very unlikely to lead to desirable placements are excluded. Good possibilities are not locked out arbitrarily and the number of branches to follow is never restricted.

By adjusting the parameter that controls the width of the search, the user can tradeoff the quality of the final solution against runtime on the computer. This is an advantage both in the beginning of the design process when only a fast upper bound on area and wire lengths is needed and in the end when the best possible result is to be obtained. In principle, the program allows a complete enumeration of the whole solution space given by the clustering tree by setting the lookahead to $d - 1$ levels (where d is the depth of the clustering tree) and by not specifying any pruning.

With a $d - 1$ level lookahead target shapes become redundant since in that case the solution space is completely explored and there is no need for bottom-up information made available by the target shapes. Similarly, if the lower area bounds of all possible shapes of clusters (sets of target shapes) are propagated up the tree, there will be no need for lookahead since the solution space will

again be completely explored. One-level lookahead and single target shapes become useful only when the solution space is partially explored. Both one-level look-ahead and single target shapes try to make information from lower levels of the hierarchy available so that decisions made on higher levels are more qualified. The information gained by the two heuristics is not identical but it is not disjoint either.

3.3. Intralevel Dependencies

In the way the algorithm was described up to now, after the placement for one level of the hierarchy was done, all the clusters on the next level of the hierarchy are processed independently in an arbitrary sequence. This can lead to undesirable results for the wire lengths as indicated in Fig. 7. This is because the center points of the parent clusters are taken as reference points for all connections between different clusters.

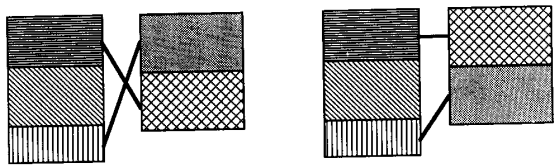
It was not attempted to find an optimal solution for this case but one that was practical from a computational point of view and would work well for most of the cases. First of all it is clear that once one cluster is placed, the center points of the cluster *elements* (which now have fixed locations) should be used to compute the edge weights of the I/O connections of other clusters (Fig. 4). This is easily achieved by attaching a "placed" bit to every cluster. When the I/O goals for a cluster are computed, all the connections to other clusters are iterated. If the connected cluster is already placed, its center point is taken as reference point. If a cluster is not yet placed, the center point of its parent is examined. During the lookahead it is possible that the examination of parent clusters is repeated recursively until an element is found whose position is already known.

This procedure introduces an ordering dependency. The earlier a cluster is placed, the stronger its influence is on the yet unplaced clusters. Therefore, clusters with larger areas are placed earlier. All clusters (whose parents are placed) are kept in a queue data structure. The ordering then is simply a matter of sorting the queue whenever a new hierarchical level is started.

3.4. Analysis

It is somewhat artificial to evaluate a placement result without routing. Nonetheless, some measure of performance must be defined in order to assess the effect of the changes to [6] introduced in this section. Two figures of merit are proposed to compare the effects with respect to layout area and wire length minimization.

The figure of merit used for the first goal is *area utilization*. It is defined as the proportion of the sum of block areas to the area of the smallest rectangle enclosing all blocks (and routing areas allocated around them). The figure of merit for the second goal is *the sum of net half perimeter* lengths (although without routing area no net could actually be implemented). The *maximal net half perimeters* are not explicitly optimized in our algorithm and only shown for comparison purposes (Table II).



(a) (b)

Fig. 7. If the center points of the parent clusters are taken as reference points for all connections between various clusters, the undesirable placement shown in (a) may be derived. The placement in (b) is, however, obtained by taking the center points of already placed clusters as the reference point for all connections to those clusters.

TABLE II
EFFECT OF TARGET SHAPE MATCHING (33-BLOCK EXAMPLE)

	original version			with target shapes		
cost function parameter	area utilization	sum of net half peri.	max net half peri.	area utilization	sum of net half peri.	max net half peri.
0.1	76.02%	10497	354	77.98%	10412	350
1	77.58%	10188	350	84.12%	10795	335
10	79.63%	10719	346	87.23%	11441	328

TABLE III
EFFECT OF PRUNING THE SEARCH TREE (33-BLOCK EXAMPLE)

prunning threshold	area utilization	sum of net half perimeters	max net half perimeters	elapsed time (normalized)
0%	87.23%	11441	328	1
50%	88.59%	10937	324	2.07
200%	90.09%	10607	339	4.67
∞	90.09%	10607	339	75.56

Even though the target shapes are better than the original heuristic, they may be misleading since in the bottom-up shape enumeration the eventual context (where the cluster will be placed) is unknown. In these cases, and of course, in those cases where target shapes include too much dead space, a lookahead can help to avoid unfavorable template choices. Table III gives some experimental results for a one-level lookahead employed in the placement of the same 33-block example. Originally, a complete search was done. The result appears in the last line of Table III. Then only branches within some percentage of the best cost value on the current level were considered on the next level of the hierarchy. This *pruning threshold* was decreased until only the best solution on the current level was pursued further on. In this case the result is the same as if no lookahead were specified.

IV. ROUTING AREA ESTIMATION

4.1. Motivation

If the placement does not include an explicit estimation of the routing area, its results will suffer from the fact that it cannot distinguish between dead space that later can be used for routing and dead space that leads to an increase in routing area because the blocks are further apart from each other.

A second reason why routing area estimation *during* the placement is necessary rather than including the routing area *after* the placement, is shown in Fig. 8. A placement that was optimized to have a rectangular shape of a given

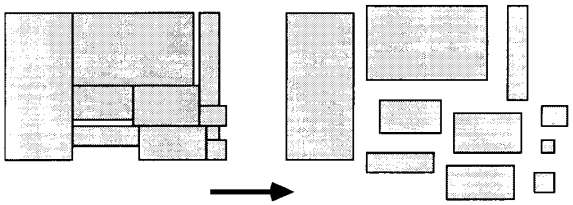


Fig. 8. A placement that was optimized to have a rectangular shape of a given aspect ratio is neither rectangular nor does it conform to the desired aspect ratio after the routing area was added.

aspect ratio is probably neither rectangular nor does it conform to the desired aspect ratio after the routing area was added.

There are some approaches that—with different degrees of sophistication—tackle the problem. In [26], [32] heuristics are applied to each block before the placement to derive a hypothetical block shape including some routing area based on the number of pins of the block. In [4] a pseudorouting of pairs of blocks is performed to derive the necessary distance between the two blocks in a row-based placement. Both solutions fail to account for connections outside of the immediate neighborhood of their terminals.

For gate arrays, Burstein *et al.* [1] introduced an algorithm that merges placement and routing in a hierarchical fashion. Because of the simple array structure, this grid-based approach is feasible. In [29] a simultaneous placement and global routing of restricted slicing structures was proposed. A method to generate the global routing at the same time as the placement, suited to the more general topologies, was described in [6]. In both cases no attempt was made to estimate the routing area based on the global routing information. Because of the representation of global routing paths as links between the center points of adjacent blocks, this task would not be straightforward.

More literature exists on the topic of routing area estimation after the placement is finished. Statistical approaches [12], [9] use a Poisson model for the generation of wires along block edges and assume an exponential distribution of wire lengths. They do not take actual pin positions or detailed connectivity information into account. Therefore, their usefulness for the problem described is questionable. Most programs perform global routing after the placement is finished and then estimate the necessary routing area [16], [11], [15].

4.2. *Top-Down Space Allocation*

Besides the hierarchical decomposition of the problem, the basic idea is to avoid a dynamic shortest path or Steiner tree determination by precomputing the paths for the finite number of templates and storing the information in the library of placement topologies (templates).

For every template and for each connection between blocks, clusters, and I/O-goals, all the channels on the shortest topological paths are marked with a probability. This probability represents the likelihood that the connection will really pass through that channel (see Fig. 9). p_{ij}^{kl} is the probability that a connection between blocks i and j will pass through the channel formed between blocks k and l. The amount of memory to store the library of parameters is acceptable [10].

The required normalized "channel"[3] width s_{kl}/w is estimated as

$$\frac{s_{kl}}{w} = t_{kl} \sum_i \sum_j p_{ij}^{kl} c_{ij}$$

where w is the design-rule dependent track to track spacing, c_{ij} is the pertinent element of the connectivity matrix, and t_{kl} a heuristic factor that accounts for track sharing. (Segments of different nets may be assigned to the same track by the channel router.) The computation can be done very fast for every possible topology and on all the hierarchical levels before the placement cost function is evaluated. In this fashion, routing area is treated equivalently to block area. It not only influences the choice of templates on the current level of the hierarchy but also sets the shape goals for the next level.

The estimation takes advantage of the information gathered about positions and connectivities of clusters (of blocks) down to that level of the tree hierarchy. Earlier in the process, space for global connections between different clusters is provided. Later, more of the internal connections within the clusters become visible.

The allocation of space along the shortest path makes the job of a global router easier but does not constrain it in doing whatever is recognized as optimal after the complete topological information produced by the placement is available. In addition, this approach is very flexible. For "over-the-block wirable" cells [30], the probabilities can be easily adjusted.

Before refining the basic idea it seems appropriate to describe how the numbers p_{ij}^{kl} and t_{kl} are derived. In general the numbers to be assigned to p_{ij}^{kl} are pretty obvious as shown in Fig. 9 since most of the time only two distinct shortest paths exist. Fine-tuning statistics can be compiled that characterize the behavior of the global and detail routers used. It is likely that different routing algorithms will yield slightly different results for the parameters. It is one of the strengths of this approach that without changes in the program it can be applied to different technologies and physical design processes.

For the non-leaf levels it is assumed that connections leave the clusters on the side that is closest to the end point of the connection. In Fig. 9 this means that a connection between blocks 0 and 2 would leave the right side of block 0 and enter the left side of block 2. It is the task of the next lower level to provide the space to get to this side. On the leaf level this is no longer possible. In this case the pin position information already needed for the determination of block orientations comes in handy. When needed, for a given block orientation additional space has

[3] "Channels" on higher levels consist of many real channels.

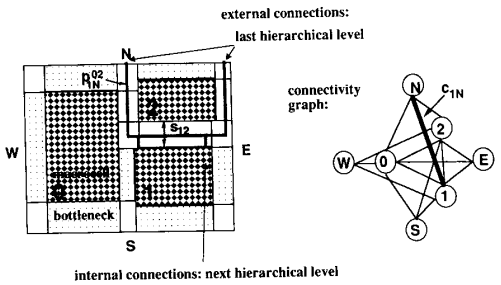


Fig. 9. This figure defines terminology used by the hierarchical area estimator. A connection between cluster 1 and I/O goal N amounts to topological paths through certain bottlenecks. In the case, $p_{1N}^{02} = 0.5$, $p_{1N}^{2E} = 0.5$, and $p_{1N}^{12} = 1.0$.

to be provided along the sides of the block to bring the wires around the block to the location that is closest to the end point of the connection.

4.3. Bottom-Up Estimation

Due to the nature of our top-down placement algorithm, area taken away from the cluster area to account for the space needed by the wiring must have been added beforehand. Instead of just assigning the sum of the areas of the children to a parent node in the clustering tree before starting the top-down traversal of the tree, a routing area estimation has to be included as well. At this stage it is not necessary to know the routes that connections take, but only the approximate *area* needed for connections has to be estimated.

The task of predicting the space needed for routing before invoking a floorplanner or placer has gained some attention recently [14], [5], [34]. Unfortunately, the reported results applicable to the macrocell layout style are not very encouraging. Errors of 20 percent (of the whole layout area, much more if the known block area is excluded) seem to be the current state-of-the-art [5], [34]. This would be unacceptable to the objective function evaluated for the different topological possibilities and would lead to very strange results.

Fortunately, a bit more information to estimate the routing area is available here. After having done the clustering on one level of the hierarchy, we know all the connections between the cluster elements and from cluster to cluster. On the lowest level of the hierarchy the number of pins along the 4 sides of a block are known as well. When the optimal target shape is chosen, a reasonable way of arranging the blocks topologically is generated as a by-product. All this information can be used to produce a routing area estimate that exactly mirrors the more sophisticated top-down routing area estimate with respect to its hierarchical decomposition.

Since the exact constellation that tries to minimize the number of connections between nonadjacent blocks/clusters is not known, a worst-case approach is taken by building a hypothetical connectivity matrix $\tilde{C}$ with identical connection strengths $\tilde{c}_i$ and $\tilde{c}_e$ for all intracluster and intercluster connections, respectively. This means that all

TABLE IV
MISMATCH BETWEEN ESTIMATED AND ACTUAL LAYOUT AREAS

number of blocks	area after placement		area after routing (core)		Δ%
10	2384	(40.4 × 59.0)	2470	(41.3 × 59.8)	+3.6
33	235.3	(15.9 × 14.8)	246.1	(16.3 × 15.1)	+4.6
9	5060	(74.9 × 67.6)	5560	(73.9 × 68.4)	-1.2

$\tilde{c}_{ij}$ are set to $\tilde{c}_i$ for $j \in 0, \cdots, k-1$ and to $\tilde{c}_e$ for $j \in N, W, S, E$:

$$\tilde{c}_i = \frac{2}{k(k-1)} \sum_i \sum_{j \in 0, \cdots, k-1} \tilde{c}_{ij}$$

$$\tilde{c}_e = \frac{1}{4k} \sum_i \sum_{j \in N, W, S, E} \tilde{c}_{ij}.$$

With this connectivity matrix and the target shape topology, the top-down wiring space allocation function is called. The exact distribution of the wiring space generated by that function is of no interest here. The only useful extraction is the total area of the space needed for routing. It is not necessary in the top-down placement process for the same topology chosen for the routing area estimate to be reasonably close. This may help of course, and really happens on lower levels of the hierarchy if the wasted area in the target shape is small. The routing area (contrary to its exact allocation) is very similar for various good topologies (i.e., topologies without too much dead space).

On the leaf level the block areas are inflated in both dimensions based on the number of pins of the block in each direction. In the final result connections between nonadjacent blocks tend to be weaker than connections between adjacent blocks. Therefore, the initial routing area estimate is reduced by some heuristic factor. If with the default value of that factor the routing area is consistently over- or underestimated, it can be adjusted by the user.

4.4. Analysis

Table IV helps to determine how well the routing area estimation predicts the area actually needed by the routing for some examples tried. The areas in the first column are given by the output of the placement program. The results in the second column reflect the final areas after the global and detailed routing were completed. The routing area needed by the ring router is not included because it is not taken into account in the routing area estimation during the placement.

V. FLOORPLANNING

5.1. Overview

This section considers floorplanning of the building blocks. We first define the problem: given 1) a set of blocks with constraints on their areas, shapes, and relative positions, 2) constraints on area and aspect ratio of the chip, and 3) a net list specifying pins to be interconnected, the floorplanning problem is to determine shapes, locations, and pin positions for the blocks so that all con-

straints are satisfied and so that the chip area and the total wire length are minimized.

Various floorplanning schemes have been devised. Wong *et al.* [33] used simulated annealing to construct slicing floorplans for flexible blocks. Otten [19] suggested a bottom-up floorplanning algorithm based on combining shape functions (width versus height curves) of flexible blocks in a slicing floorplan structure and propagating the composite shape function recursively up a clustering tree. After shape function for the entire chip is determined, a boundary point on the shape function satisfying aspect ratio, width, or height goal is chosen and this information is passed top-down until shapes of the leaf blocks are determined. Wimer *et al.* [31] proposed a branch and bound algorithm for computing the composite shape function for nonslicing structures. Maling *et al.* [17] proposed a floorplan design algorithm based on enumerating all valid rectangular duals of a connectivity graph for the chip and choosing a rectangular dual with the least area which satisfies shape and position constraints. Mogaki *et al.* [18] used linear programming techniques to determine shapes of the flexible blocks subject to constraints on their relative positions.

In the BEAR layout system, we decompose the floorplanning problem into several phases, each of which is more clearly defined and is simpler to solve [21]. Initially, a clustering tree with blocks having their preferred aspect ratios is derived. During the top-down traversal of the tree, the geometry cost function is changed so that the penalty associated with the mismatch between the flexible blocks and the rooms (to which these blocks are assigned) is reduced. In the end, flexible blocks are resized (subject to their shape constraint functions) so that they fit "best" in their assigned rooms. When a flexible block is resized, its pins on the sides being shrunk are pushed closer by a scaling factor. Similarly, its pins on the sides being stretched are pulled apart. This initial floorplanning is followed by the global routing process which defines densities of the routing areas. After global routing, a global spacing procedure assures that all bottleneck tiles have capacities equal to or exceeding their corresponding densities.

A global shape optimization phase follows next: given the initial floorplan and constraints on block shapes, the shape optimizer determines locations and shapes of the blocks so that the chip area is minimized. At present, the shape optimizer does not optimize locations of the pins on blocks or orientations of blocks.

One- and two-dimensional shape optimization algorithms have been implemented. The one-dimensional algorithm iteratively computes new dimensions for flexible blocks in order to reduce the chip dimension in the user-specified *resize direction* (horizontal or vertical). This algorithm does not change chip dimension in the direction orthogonal to the resize direction. The two-dimensional algorithm iteratively reduces the layout area by picking up a block with the largest resize possibility and resizing it.

5.2. Basic Definitions

The individual blocks are either *rigid* or *flexible*. Rigid blocks cannot change shape during the floorplanning process and represent blocks that have already been layed out and have fixed shapes and fixed I/O interfaces. Flexible blocks can change shape and represent blocks which are only partially designed or blocks which are procedurally synthesized by parametrizable module generators or silicon compilers. The manner in which a flexible block is reshaped is inferred from the integrated circuit design style which is used to lay out the block. For example, the *gate array* design style only allows discrete change in either dimension of the block whereas the *standard cell* design style allows discrete change in one dimension and continuous change in the other. The *general cell* design style permits continuous change in both dimensions of the block. (See [23] for a description of various design styles.)

For a block layed out in gate array or standard cell styles we assume a finite set of (x, y) pairs where x and y are block dimensions, and for a block layed out in general cell style we assume a hyperbolic shape function.[4] The floorplanner can accurately estimate shape function curves for various standard cell blocks by examining the block netlists as described in [20]. We assume that each flexible block has a *preferred* aspect ratio (e.g., aspect ratio which leads to minimum block area).

The entire area of a layout is covered with rectangles referred to as *tiles*. There are two kinds of tiles: *solid tiles* which represent blocks and *space tiles* which represent empty space for routing between the blocks. Given a placement of building blocks, we define two tile planes: the *horizontal tile plane* where all space tiles are maximal horizontal strips and the *vertical tile plane* where all space tiles are maximal vertical strips (see Fig. 10). A space tile is called *bottleneck* if both sides are covered by the sides of its adjacent space tiles. We store global routing information on the bottleneck tiles. *Block adjacency graphs* [7] need not be constructed explicitly since the adjacency of the blocks can be obtained efficiently via bottleneck tiles.

The block adjacency graphs can be used to calculate the extents of the chip. The longest or *critical* paths through the horizontal and vertical block adjacency graphs determine width and height of the layout. Vertices of the horizontal block adjacency graph represent blocks, and arcs represent horizontal bottleneck tiles. Weights on vertices are horizontal dimensions of the corresponding blocks, and weights on arcs are densities of associated bottleneck tiles. The vertical block adjacency graph is defined similarly. We use bottleneck densities rather than bottleneck capacities since longest paths computed using bottleneck densities give more accurate estimations of the post-layout chip dimensions. In addition, because the global routing information is incrementally updated and bottleneck

[4]In this paper we assume that block area remains constant as block aspect ratio is changed. This assumption must be revised in order to come up with a more realistic model for general cells.

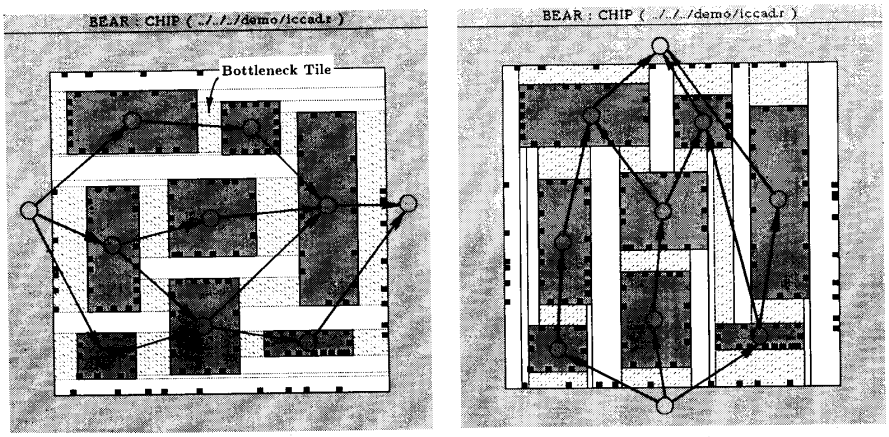


Fig. 10. This figure shows the horizontal and the vertical tile planes where bottleneck tiles are shaded. The horizontal and the vertical block adjacency graphs are drawn. Vertices of the graphs represent blocks and arcs represent bottleneck tiles.

densities recomputed from one iteration to the next, longest paths through the layout surface remain accurate and representative of the final chip dimensions.

5.3. *Shape Optimization Algorithm*

We denote the estimated width and height of the chip at the jth iteration W and H, respectively. If block i with width w^i, height h^i, and area a^i does not belong to the longest path in the current optimization direction, it will have some freedom to move or deform in that direction without enlarging the chip area. Considering the X-direction, the legal X-slack of this block $x^i_{\text{legalSlack}}$ is determined as follows: assume that the length of the longest path from the left boundary of the chip to the left boundary of the block is l^i and the length of the longest path from the right boundary of the block to the right boundary of the chip is r^i. Then, the X-span of the block (horizontal range where the block can be placed without overlapping other blocks or causing overflows in the neighboring bottleneck tiles) is given by

$$x^i_{\min} = l^i$$

$$x^i_{\max} = W - r^i.$$

The lower left corner of the block may be positioned between $x^i_{\min}$ and $x^i_{\max} - w^i$ or its width may be increased by an amount $x^i_{\text{slack}} = x^i_{\max} - x^i_{\min} - w^i$. The two operations can be combined without enlarging the chip dimension in the X-direction. Now, suppose that the block height has a lower bound h^i_{lower} and its width has an upper bound of w^i_{upper}. Then

$$x^i_{\text{legalSlack}} = \min\left(x^i_{\text{slack}}, w^i_{\text{upper}} - w^i, \frac{a^i}{h^i_{\text{lower}}} - w^i\right).$$

Similarly, the legal Y-slack $y^i_{\text{legalSlack}}$ of block B^i is defined as a function of the upper bound h^i_{upper} on its height and the lower bound w^i_{lower} on its width (see Fig. 11).

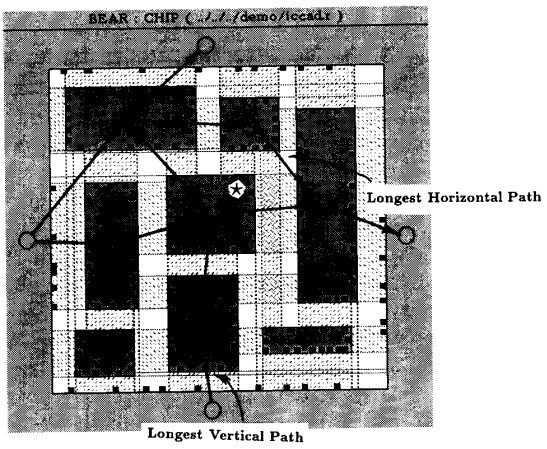
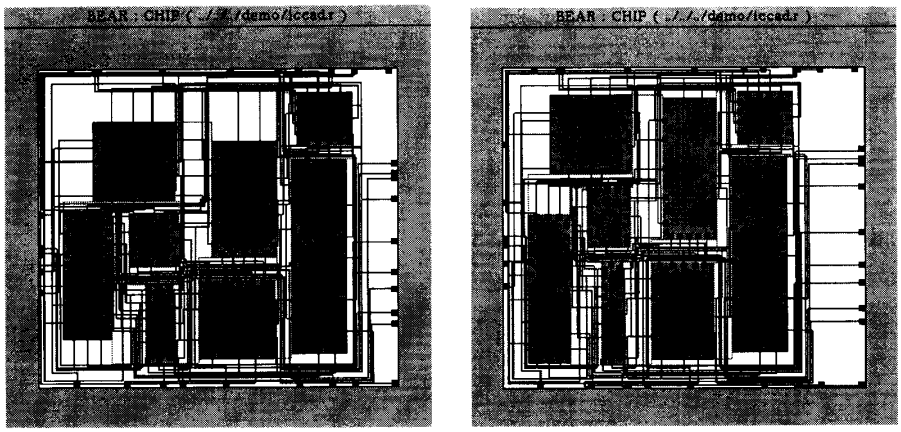


Fig. 11. The flexible block marked with a ★ lies on the longest vertical path ($y^{\star}_{\text{legalSlack}} = 0$) but has a nonzero legal slack in the horizontal direction. By resizing this block by $\gamma x^{\star}_{\text{legalSlack}}$ the chip height decreases but the chip width will remain unchanged.

We outline the procedure for two-dimensional shape optimization. We initially compute the longest paths through the layout in both X- and Y-directions. Next, we find a block B_i which lies on the longest path in one direction (e.g., Y-direction) and has the largest *legal* slack in the other direction (e.g., X-direction). This block may be laterally shifted (in X-direction) and/or resized by $\Delta w^i = \gamma x^i_{\text{legalSlack}}$ where $0 < \gamma \leq 1$ is a user specified parameter bounding the maximum change in block dimensions per iteration. We have observed that $\gamma = 0.7 - 0.8$ leads to uniform and fast convergence. The new block dimensions are $\hat{w}^i = w^i + \Delta w^i$ *and* $\hat{h}^i = a^i/\hat{w}^i$ if block B_i is a general cell. If block B_i is generated by gate array or standard cell layout style, $\hat{w}^i$ and $\hat{h}^i$ will be rounded to the closest (x, y) pair in the block shape function. The new position of block is adjusted in order to distribute "free"



(a) (b)

Fig. 12. This figure shows the layout result for an 8-block example without (a) and with (b) shape optimization.

area among the surrounding bottleneck tiles according to their wiring demands. The global spacer may be called to assure that no surrounding bottleneck tiles will overflow as the result of block resizing. The global routing around the block is locally updated. After block B_i is resized, the chip height is usually decreased[5] but the chip width remains relatively unchanged. In practice, the optimization process is monotonically nonincreasing with respect to chip area.

In general, the stopping criterion is that longest paths only contain fixed size blocks (or maximally warped flexible blocks) or contain flexible blocks which lie on the longest paths in both X- and Y-directions. To speed up the algorithm, the process is terminated if in the last k iterations the area reduction has been less than some small percentage of the total chip area *or* if the largest legal slack in the X- and Y-directions is less than a user specified bound.

For small circuits (less than 12–15 blocks), this algorithm is very efficient and gives excellent results. For larger circuits, because of the unpredictability of the changes made to the underlying topology, the algorithm may require many iterations. Therefore, we devise a two-pass one-dimensional shape optimization algorithm (an X-direction pass followed by a Y-direction pass) which converges to a good solution faster. This is because by fixing the resize direction we avoid making drastic changes to the original topology. However, as is expected, the two-dimensional shape optimization often gives better results than the two one-dimensional shape optimizations.

5.4. Analysis

The two-dimensional shape optimization problem is more general than the two-dimensional compaction problem since both positions and shapes of blocks are varied to minimize the layout area. True two-dimensional compaction has been attempted with exponential complexity algorithms. It is proven that this problem is *NP*-complete [24], [25] and so is the shape optimization problem.

The complexity of each iteration of the shape optimization algorithm is $O(|V|)$ where $|V|$ is the number of the blocks. This is because the longest path computation using topological sorting of the nodes in a directed acyclic graph is $O(|E|)$ and the *block adjacency graphs* are directed acyclic graphs with $|E| = O(|V|)$ (see [8]).

The shape optimizer was run on several layouts generated by the placement algorithm. On average, the layout areas were reduced by 10 percent and the total wire lengths (after detailed routing) were cut by 5 percent. Fig. 12 shows the layouts for a 8-block example with and without shape optimization. All blocks were assumed to be flexible and their aspect ratios were allowed to change in the range from 0.5 to 2. The area and wire length reductions are 11.2 and 5.1 percent, respectively. The stopping criterion were an area reduction less than 1 percent in the last 5 iterations or largest legal slack less than 8 μm for all flexible blocks in the chip.

V. Sensitivity

Since CAD programs do not produce a global optimum solution for most problems, incremental changes in the problem description should not yield solutions that are radically different from the original ones. Small changes in the input occur quite frequently. To overcome functional problems, connections are changed, added, or removed. To optimize timing of the circuit or to lower heat dissipation in critical areas, sizes of transistors are changed. If the CAD program is too sensitive to these changes and produces a totally different layout, this might defeat the purpose for input modifications, possibly causing further changes in the input and requiring many iterations until the solution is stable again (Fig. 13). On the other hand the algorithm should be able to react to signif-

[5]It may not be reduced due to the existence of multiple longest paths through the layout surface.

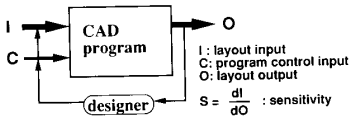


Fig. 13. A good CAD program should be sensitive to layout input (I) and control (C). However, it should not be too sensitive to input changes otherwise the solutions produced by the program may be unstable.

icant changes in the input I, otherwise it could not find nearly optimal solutions. Furthermore, it should be sensitive to the control input C, otherwise a designer could not influence the program to consider his particular requirements.

Maintaining the global layout without sacrificing the ability to optimize it, is possible by taking the original clustering tree again for the changed input data. The clustering tree still allows the placement to adapt to changes in the design in various ways, whereas a binary slicing tree would already be quite restrictive. The sensitivity can be controlled in still another way: when looking for the minimum of the objective function, a better solution is only accepted if its cost is *significantly* lower than that of the best solution up to date. What is significant, is determined by the user, according to how important this issue is for him. This introduces a bias towards the solutions produced earlier, so the placement templates are enumerated in a sequence corresponding to their general "desirability."

Changes in the placement due to the addition of routing area are minimized by the simultaneous placement and routing area estimation described in Section IV. To be able to evaluate the effectiveness of various measures to decrease the sensitivity S, the formula for S has to be changed to $S = \Delta O/\Delta I$ and input and output changes have to be clearly defined. ΔI^C can be defined as the proportion of nets that are changed, added, or deleted to the total number of nets. (If the changes are small, the denominator can be treated as being constant.) Similarly, ΔI^B can be defined as the proportion of area changes to the sum of all block areas. To avoid this, positive and negative area changes cancel each other out; their absolute values are added to ΔI^B with a positive sign.

Since the program under consideration is a placement program, ΔO is measured in terms of deviations from the original block positions: for the origin always lying in the lower left-hand corner of the chip, ΔO can be defined as

$$\Delta O = \sum_i \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right)$$

where (x_i, y_i) is the original position of block i and $(\hat{x}_i, \hat{y}_i)$ is the block position after the changes. The argument to use the squares of all distances is to make a few large movements count more than many small ones.

It is practical to split up the sensitivity into two parts: $S^C (= \partial O/\partial I^C)$, the sensitivity for changes in the connectivity input and $S^B (= \partial O/\partial I^B)$, the sensitivity for changes in the block area input.[6] Symbolically this can be written as

$$dO = S^C \times dI^C + S^B \times dI^B$$

where dI^C and dI^B represent the respective input changes.

Table V shows the effects of sensitivity reduction mechanisms described earlier in this section. Results in the first row were obtained by changing two connections (ΔI^C) and results in the second row by changing two block shapes (ΔI^B).

TABLE V
EFFECT OF SENSITIVITY REDUCTION MECHANISMS (33-BLOCK EXAMPLE)

clustering tree	no constraint		original tree	
significance threshold for changes	0.00	0.00	0.01	0.05
ΔO (1 net deleted, 1 changed)	596	0	0	0
ΔO (2 blocks resized)	2988	2988	2154	504

TABLE VI
RESULTS ON PRIMBBL2 WITH 33 BLOCKS AND 203 NETS

system	chip area	wire length	vias
BEAR	28.47	633494	897
MOSAICO	29.01	650009	1173
VITAL	31.17	865712	1029
Seattle Silicon	28.63	762000	1235
Delft P&R	26.57	615104	925

TABLE VII
RESULTS ON PRIMBBL1 WITH 10 BLOCKS AND 123 NETS

system	chip area	wire length	vias
BEAR	2.83	131244	798
MOSAICO	3.16	151824	813
VITAL	3.12	134599	763
Seattle Silicon	2.94	125000	948
Delft P&R	2.60	151656	967

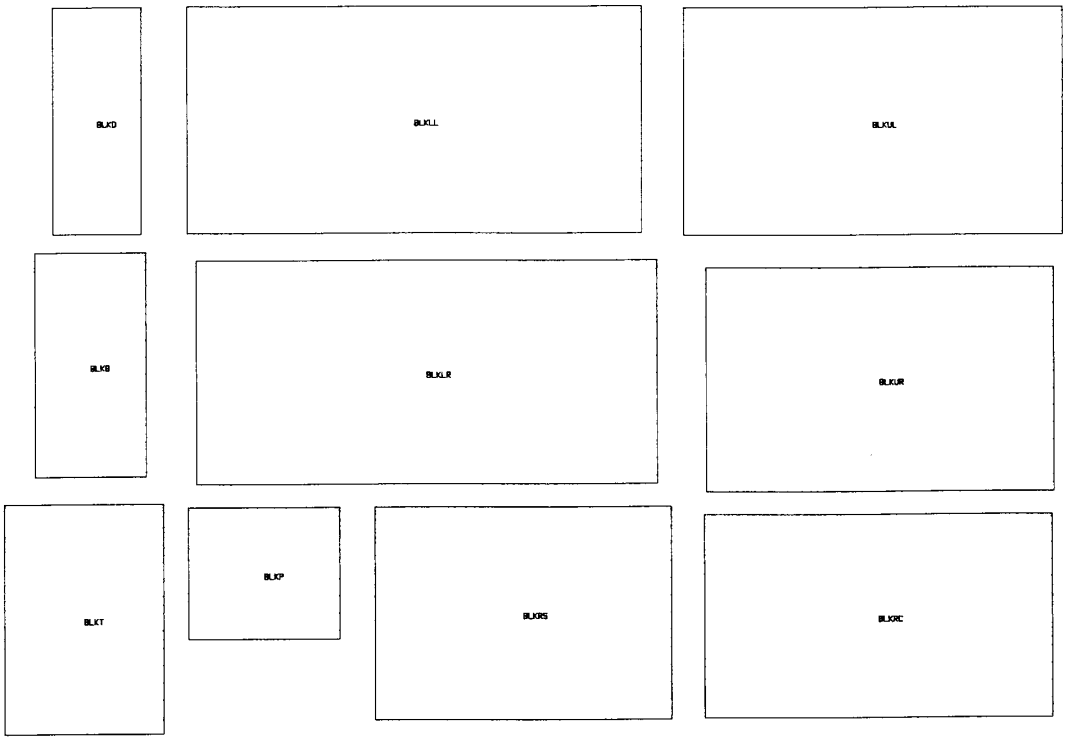
VII. RESULTS

The BEAR system has been tested with several examples including the two benchmark examples used at the 1988 International Workshop on Placement and Routing. In Tables VI and VII we compare our results with those reported at the workshop.[7] The time needed to obtain the placements by the BEAR placer was 131 s for PrimBBL1 and 762 s for PrimBBL2 (elapsed time on a VAX 8880, workload 10.5—12.5). The layouts produced by BEAR are shown in Figs. 14 and 15.

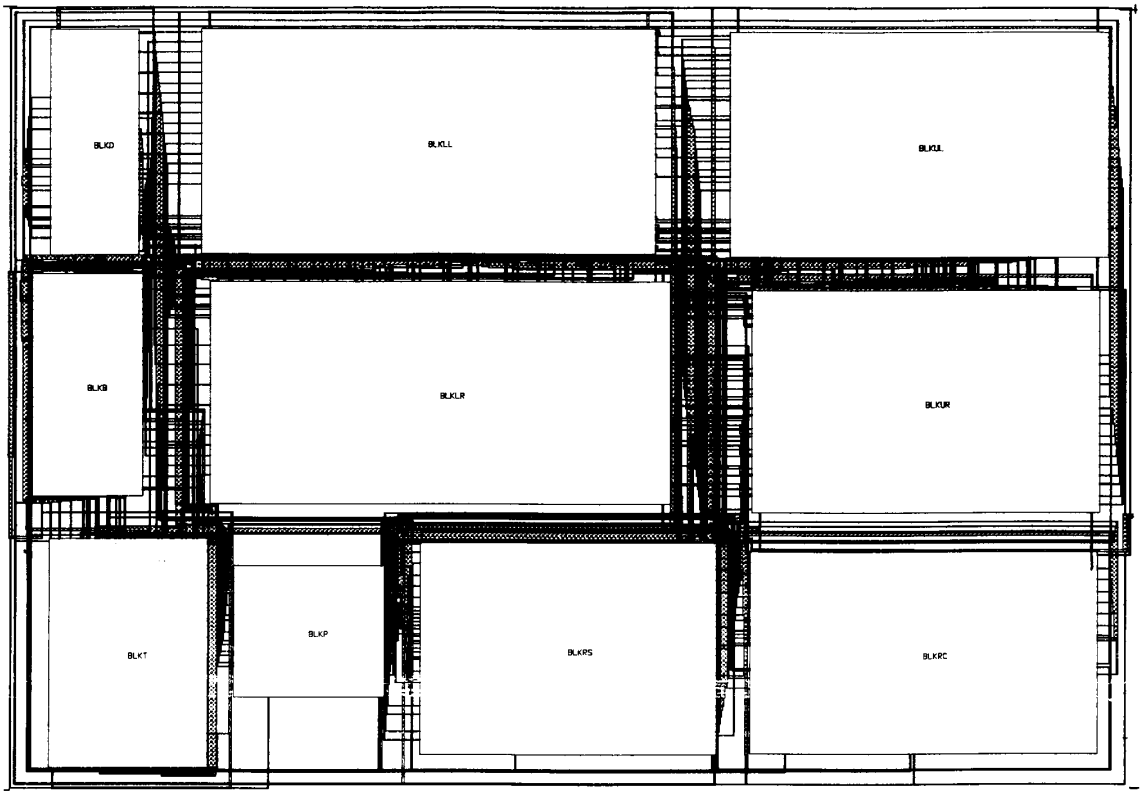
We were also able to complete the routing with BEAR on placements for PrimBBL1 obtained by BBL [4], MOSAICO [2], and ATLAS [27]. By factoring out the influence of different routing systems, the placement evaluation becomes more meaningful. All the numbers in Table VIII are relative to the best placement produced by BEAR.

[6]Other sensitivities—like the sensitivity for block shape changes—could be readily incorporated as well, but are not considered in this paper.

[7]*VITAL* treats P&G nets as signal nets. *Seattle Silicon* routes I/O connections only to the chip boundary and not to the pins. *Delft* does the placement manually.

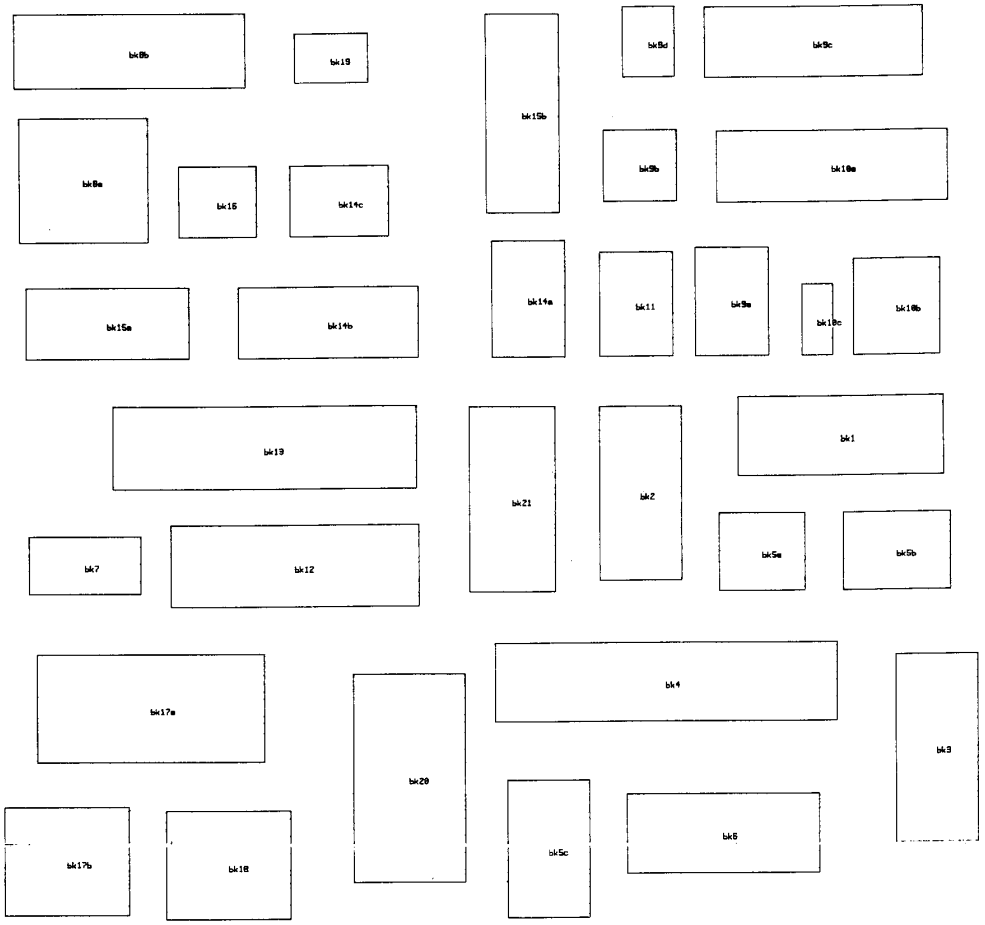


(a)



(b)

Fig. 14. This figure shows BEAR (a) placement and (b) routing for PrimBBL1 benchmark.



(a)

Fig. 15. This figure shows BEAR (a) placement and (b) routing for PrimBBL2 benchmark.

VIII. Future Work

A possibility to improve the block shape information that is propagated up is to replace the single target shape with a set of target shapes (a shape function). In that case no more lookahead steps would be necessary. The top-level goal shape could be chosen from a finite set. Shape functions can be combined efficiently for slicing structures [19]. For nonslicing structures it seems that there is no way to find the composite shape function other than enumerating all the possibilities.[8]

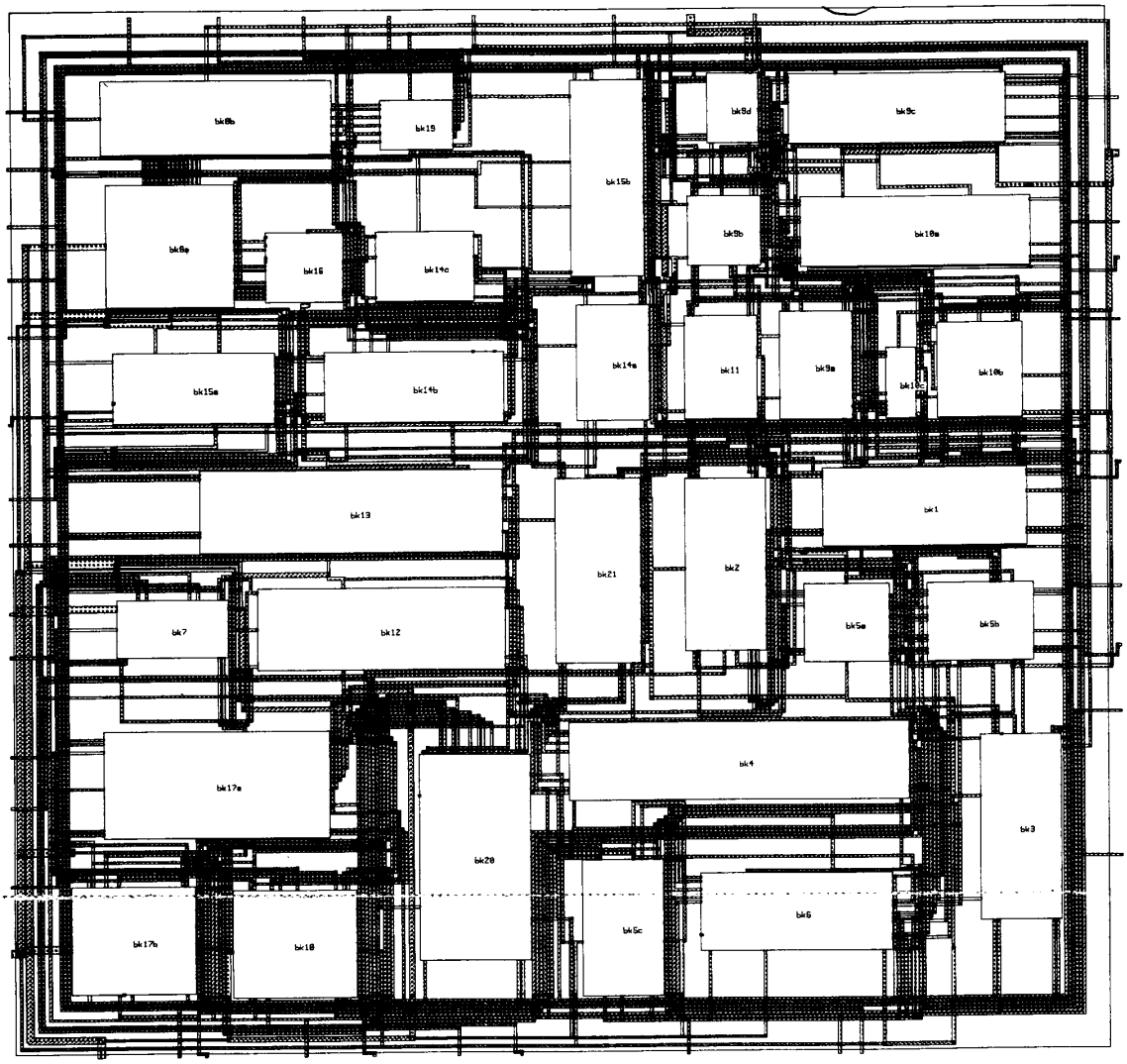
The input to a general placement program for building blocks may include L-shaped blocks. These blocks are generated by block abutments or custom layout. The inclusion of L-shaped blocks in the BEAR placement is easy due to the template enumeration approach adopted. New templates representing L-shaped rooms as well as rectangular rooms have to be added.

Automatic pin assignment for flexible blocks is needed. During the initial floorplanning phase where circuit topology is yet unknown only a rough pin assignment (e.g., side of the block pins should be placed on) will be performed. Later, during the shape optimization phase where circuit topology and connection paths for all nets are known, more detailed pin assignment based on available global routing information will be performed. We optimize pin assignment per net, per block, or globally.

Automatic I/O pad assignment in BEAR (i.e., assigning off-chip I/O's to the pads prior to block placement) will be addressed as well. This procedure may be timing-driven, satisfying path delay constraints or connectivity-driven minimizing total wire lengths.

Efforts are under way to incorporate timing constraints in the BEAR floorplanning. The floorplanning process usually has a dominant effect on the circuit performance. The timing requirements are stated as required arrival times at I/O pins of blocks or as required delay from a

[8] In [28] it was proven that finding the optimal orientations of blocks for a given floorplan is *NP*-complete for nonslicing structures. Because the problem of finding the shape function—lower area bound of all possible rectangles—of a cluster involves finding the optimal orientations of the cluster elements, that problem is clearly *NP*-complete as well.



(b)

Fig. 15. (*Continued*)

TABLE VIII
RELATIVE COMPARISON OF DIFFERENT PLACEMENTS

system	aspect ratio (x/y)	chip area	wire length	vias
BEAR	0.72	1.000	1.000	1.000
BBL	0.70	1.077	1.179	N/A
BEAR	0.96	1.001	1.145	1.028
MOSAICO	1.04	1.093	1.206	1.054
ATLAS	1.11	1.112	1.345	1.195

primary input to a primary output of the chip. Area and performance parametrizable module generators can add a new degree of flexibility (and complexity) by giving the floorplanning process the ability to tradeoff block delay for block area in order to optimize layout and circuit performance simultaneously.

REFERENCES

[1] M. Burstei, S. J. Hong, and R. Pelavin, "Hierarchical VLSI layout: Simultaneous placement and wiring of gate arrays," in *Proc. VLSI 1983*, pp. 45–60, 1983.

[2] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macrocells," in *Dig. Tech. Papers, IEEE Int. Conf. Computer-Aided Design*, pp. 30–33, 1986.

[3] N. P. Chen, C. P. Hsu, E. S. Kuh, C. C. Chen, and M. Takahashi, "BBL: A building block layout system for custom chip design," in *Dig. Tech. Papers, IEEE Int. Conf. Computer-Aided Design*, pp. 40–41, 1983.

[4] C. C. Chen and E. S. Kuh, "Automatic placement for building block layout," in *Dig. Tech. Papers, IEEE Int. Conf. Computer-Aided Design*, pp. 90–92, 1984.

[5] X. Chen and M. L. Bushnell, "A module area estimator for VLSI layout," in *Proc. 25th Design Automation Conf.*, pp. 54–59, 1988.

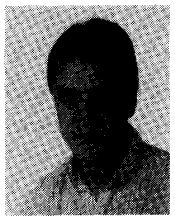
[6] W. Dai and E. S. Kuh, "Simultaneous floor planning and global routing for hierarchical building block layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 828-837, 1987.
[7] W. Dai, M. Sato, and E. S. Kuh, "A dynamic and efficient representation of building block layout," in *Proc. 24th Design Automation Conf.*, pp. 376-384, 1987.
[8] W. Dai and E. S. Kuh, "Global spacing of building block layout," *VLSI 1987*, pp. 161-173, 1987.
[9] A. A. ElGamal and Z. A. Syed, "A stochastc model for interconnections in custom integrated circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 883-893, 1981.
[10] B. Eschermann, "Hierarchical placement for macrocells with simultaneous routing area allocation," Mem. UCB/ERL M88/49, Univ. Calif., Berkeley, 1988.
[11] C. Fowler, G. D. Hachtel, and L. Roybal, "New algorithms for hierarchical place and route of custom VLSI," in *Dig. Tech. Papers, Int. Conf. Computer-Aided Design*, pp. 273-275, 1985.
[12] W. R. Heller, W. F. Mikhail, and W. E. Donath, "Prediction of wiring space requirements for LSI," *Design Automation Fault-Tolerant Computing*, pp. 117-144, 1978.
[13] M. Khellaf, "On the partitioning of graphs and hypergraphs," Ph.D. dissertation, Dept. IEOR, Univ. Calif., Berkeley, 1987.
[14] F. J. Kurdahi and A. C. Parker, "PLEST: A program for area estimation of VLSI integrated circuits," in *Proc. 23rd Design Automation Conf.*, pp. 467-473, 1986.
[15] D. L. LaPotin and S. W. Director, "Mason: A global floor-planning tool," in *Dig. Tech. Papers, Int. Conf. Computer-Aided Design*, pp. 143-145, 1985.
[16] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," in *Proc. 16th Design Automation Conf.*, pp. 1-10, 1979.
[17] K. Maling, S. Mueller, and W. Heller, "On finding the most optimal rectangular package plan," in *Proc. 19th Design Automation Conf.*, pp. 663-670, 1982.
[18] M. Mogaki, C. Miura, and H. Terai, "Algorithm for block placement with size optimization technique by the linear programming approach," in *Proc. Int. Conf. Computer Design*, pp. 80-83, 1987.
[19] R. H. J. M. Otten, "Efficient floorplan optimization," in *Proc. Int. Conf. Computer Design*, pp. 499-502, 1983.
[20] M. Pedram and B. Preas, "Accurate prediction of physical design characteristics for random logic," to be published.
[21] M. Pedram, "Hierarchical floorplanning for macrocells," Memo. UCB/ERL, Univ. Calif., Berkeley, 1989.
[22] B. T. Preas and P. G. Karger, "Automatic placement—A review of current techniques," in *Proc. 23rd Design Automation Conf.*, pp. 622-629, 1986.
[23] B. T. Preas, "Introduction to physical design automation," in *Physical Design Automation of VLSI Circuits*, B. T. Preas and M. Lorenzetti, Eds. Menlo Park, CA: Benjamin/Cummings, 1988, pp. 14-18.
[24] S. Sastry and A. Parker, "The complexity of two-dimensional compaction of VLSI layout," in *Proc. ICCC-82*, pp. 402-406, 1982.
[25] M. Schlag, Y. Z. Liao, and C. K. Wong, "An algorithm for optimal two-dimensional compaction of VLSI layouts," *Integration*, pp. 179-209, 1983.
[26] C. Sechen and A. Sangiovanni-Vincentelli, "The Timber Wolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510-522, 1985.
[27] L. Sha and R. W. Dutton, "An analytical algorithm for placement of arbitrarily sized rectangular blocks," in *Proc. 22nd Design Automation Conf.*, pp. 602-607, 1985.
[28] L. Stockmeyer, "Optimal orientation of cells in slicing floorplan designs," *Inform. Contr.*, vol. 57, pp. 91-101, 1983.
[29] A. A. Szepieniec, "Integrated placement/routing in sliced layouts," in *Proc. 23rd Design Automation Conf.*, pp. 300-307, 1986.
[30] K. Ueda, H. Kitazawa, and I. Harada, "CHAMP: Chip floor plan for hierarchical VLSI layout design," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 12-22, 1985.
[31] S. Wimer, I. Koren, and I. Cederbaum, "Optimal aspect ratios of building blocks in VLSI," in *Proc. 25th Design Automation Conf.*, pp. 66-72, 1988.
[32] G. J. Wipfler, D. A. Mlynsk, and H. Hillner, "An automatic placement procedure for integrated circuits," in *Proc. Int. Symp. Circuits Systems*, pp. 13-16, 1985.
[33] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. 23rd Design Automation Conf.*, pp. 101-107, 1986.
[34] G. Zimmermann, "A new area and shape function estimation technique for VLSI-layout," in *Proc. 25th Design Automation Conf.*, pp. 60-65, 1988.

*

Wayne Wei-Ming Dai (S'84-M'88) received the B.A. degree in computer science and the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley, in 1983 and 1988, respectively.

He worked at AT&T Bell Laboratories, Murray Hill, NJ and NEC Research Laboratories, Kanagawa, Japan, during the summers of 1985 and 1987, respectively. He is currently an Assistant Professor of Computer Engineering at the University of California, Santa Cruz. His research interests include computer-aided synthesis of VLSI circuits, graph theory, and computational geometry.

*



Bernhard Eschermann received the Diplom-Ingenieur degree in electrical engineering from the University of Karlsruhe, West Germany, and the M.S. degree in electrical engineering from the University of California, Berkeley, in 1987 and 1988, respectively. He is currently working towards the Ph.D. degree in computer science at the University of Karlsruhe, West Germany.

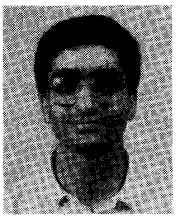
His current research interests include VLSI layout and logic design for testability.

*

Ernest S. Kuh (S'49-M'57-F'65) received the B.S. degree from the University of Michigan, Ann Arbor, the S.M. degree from the Massachusetts Institute of Technology, Cambridge, and the Ph.D. degree from Stanford University, Stanford, CA, in 1949, 1950, and 1952, respectively.

From 1952 to 1956, he was a member of the Technical Staff, Bell Telephone Laboratories, Murray Hill, NJ. In 1956, he joined the Electrical Engineering and Computer Sciences Department, University of California at Berkeley. From 1968 to 1972, he served as chair of the department, and from 1973 to 1980, he served as Dean of the College of Engineering. He is currently a professor of electrical engineering at the University of California, Berkeley.

*



Massoud Pedram received the B.S. degree in electrical engineering from California Institute of Technology in 1986 and the M.S. degree in electrical engineering and computer science from the University of California, Berkeley in 1988. He is currently working towards the Ph.D. degree at the University of California, Berkeley.

He held summer positions at the Caltech high energy physics labs during 1985 and 1986. During 1987 and 1988, he worked as a research intern at Xerox, Palo Alto Research Center. His present research interests include area estimation techniques, floorplanning, and system partitioning.