ESP: Placement by Simulated Evolution

RALPH M. KING AND PRITHVIRAJ BANERJEE, MEMBER, IEEE

Abstract—**ESP (Evolution-based Standard cell Placement) is a new program package designed to perform standard cell placement including macro-block placement capabilities. It uses the new heuristic method of simulating an evolutionary process in order to minimize the cell interconnection wire length. While achieving comparable results to popular Simulated Annealing algorithms, ESP usually requires less CPU time. A concurrent version designed to run on a network of loosely coupled processors, such as workstations connected via ETHERNET, has also been developed. For medium to large circuits (>250 cells per processor) Concurrent ESP achieves linear speedup.**

Index Terms—**Standard cell placement, new heuristics, iterative improvement, stochastic algorithm, evolution, mutation, distributed processing.**

I. INTRODUCTION

I.1. The Placement Problem and Previous Algorithms

Given a set of standard cells of common height and variable width and the interconnections between the cells, the objective of a standard cell placement algorithm is to arrange cells in an integrated circuit layout such that it permits automatic routing of the interconnections while satisfying one or more possibly conflicting goals. Examples of such goals are minimizing layout area, maximizing circuit performance, minimizing timing delays on critical nets, etc. Often such goals are difficult to cast into an objective function that can be evaluated by a computer, hence a more restricted objective must be substituted. A discussion of various objective functions that have been proposed by researchers in the past is given in the next subsection. It has been shown that the placement problem is NP-hard [1], [2]. Since large numbers of cells are involved, an optimum solution cannot be obtained. Hence, some heuristics are needed to prune the immense search space and to find a placement close to the global optimum.

Placement methods fall into two classes [3]–[5]: *Constructive and Iterative*. Constructive methods produce a complete placement (all cells have assigned positions) based on a partial placement (some or all cells do not have assigned positions). Constructive algorithms are typically divided into the following classes: (1) cluster growth [4], (2) partitioning based placement [6]–[8], (3) global techniques like quadratic assignment and convex function optimization [9], [10], and (4) branch-and-bound techniques [4]. Iterative methods attempt to improve a complete placement by producing a new better placement. Constructive placement algorithms are normally used for initial placement, and are usually followed by iterative algorithms. Within one iteration, certain cells are selected and moved to alternate locations. If the resulting configuration is better than the old one, the new configuration is retained; otherwise the previous configuration is restored. Some of the conventional iterative improvement techniques include pairwise interchange [11], force-directed interchange, and force-directed relaxation [12].

The previous iterative techniques accept trial placements only if the objective function does not increase. This characteristic may cause an algorithm to get stuck in a local minimum rather than finding the global optimum. The Simulated Annealing (SA) technique proposed by Kirkpatrick *et al.* [13] is a general combinatorial optimization technique that uses a probabilistic hill-climbing method to get around this problem. However, this algorithm has several drawbacks, mainly its tremendous CPU time requirements and the need for an efficient annealing schedule [14]. Simulated Annealing (SA) has been proven to converge to a globally optimal result given an arbitrary amount of computation time. If less CPU time is provided, it will produce near-optimal solutions. The SA technique has been successfully used in the standard cell placement problem in the TimberWolf placement and routing package [15]–[17]. Other implementations of SA applied to standard cell placement have been reported as well [18].

In this paper, we propose a new heuristic for standard cell placement that combines the features of iterative improvement and constructive placement with the ability to avoid getting stuck at local minima using a stochastic approach. The heuristic is based on an analogy between the natural selection process in biological environments and the method of solving engineering problems by iterative improvements; we call this approach Simulated Evolution. An initial implementation of the evolution-based algorithm was first reported by us last year at the Design Automation conference [19]. The algorithm has been implementated in an Evolution-based Standard cell Placement (ESP) program. For compatibility reasons, the current version of ESP accepts the TimberWolf input data format [16]. The ESP program also supports most options available in TimberWolf such as pad and macroblock placement.

Recently, another placement algorithm called Genetic

Manuscript received May 27, 1988; revised October 20, 1988 and November 8, 1988. This work was supported by the Semiconductor Research Corporation under Contract 87-DP-109. The review of this paper was arranged by Associate Editor R. H. J. M. Otten.

The authors are with the Computer Systems Group, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

IEEE Log Number 8825914.

Placement has been proposed [20]. It is based on similar general ideas as our algorithm, however, it differs significantly in the actual procedure used. Our algorithm is *not* a genetic approach and also differs significantly from Simulated Annealing. The differences are described in Section III.10.

I.2. Review of Cost Functions

The effectiveness of a given heuristic and the evaluation of the resulting placement quality strongly depend on the cost function used. Since the global objective of a placement package is to produce a routable layout, the cost function used should model the routability of the chip as realistically as possible.

The most general form of an interconnection topology is the Steiner tree [21], which can be approximated by the half perimeter of the smallest rectangle enclosing the pins of the net [11], [22]. Other cost functions and their applications can be found in [6], [23]. The cost function used by ESP is the minimal wire length using the bounding box approximation for each net. This allows the direct comparison of our results with placements computed by programs such as TimberWolf [16].

I.3. Outline of this Paper

Section II gives an introduction to the general concept of Simulated Evolution. The structure of the ESP algorithm is outlined in Section III. Performance results of the program using several example circuits are presented in Section IV. Section V discusses Concurrent ESP, an implementation of the algorithm designed for a distributed computing environment. Subsequently, workload partitioning issues and the general implementation of Concurrent ESP are discussed in Section VI. Section VII presents performance results of Concurrent ESP. Conclusions and future work are outlined in the last section.

II. The Concept of Simulated Evolution

The novel cell placement algorithm proposed in this paper is based on an analogy with the natural selection process in biological environments and uses the notions of *evolution* and *mutation*. The biological solution to the selection process is the *evolution* from one generation to the next one by eliminating ill-suited designs and keeping near-optimal ones. Every constituent of each generation has to constantly prove its functionality under the current conditions in order to remain unaltered. The purpose of this process is to gradually create stable structures which are finally perfectly adapted to the given constraints. In many cases, the engineering environment (such as the cell placement problem) is even better suited to the evolutionary process than the natural one since the given constraints remain constant.

Nature also has a way of preventing the development of species from getting stuck at local optima by using the concept of *mutation*. Mutation can be defined as a pseudorandom process which alters the characteristics of the design in an unpredictable way. The altered design is again subjected to the evolutionary process which determines its survival. The mutation rate is normally very small compared to the evolution rate.

The idea of applying natural selection processes to engineering problems may sound unfamiliar at first but there are several very promising aspects associated with that idea. A similar concept was proposed about twenty-five years ago in a paper on logic minimization [24].

The classes of problems which seem suitable for applying evolutionary algorithms are mainly those that are not easily solved in a closed form, i.e., NP-hard and NP-complete problems. Since the computation of an exact solution to these problems is normally not feasible except for very small problem sizes, usually some heuristics are applied to reduce the search space and generate an approximate solution.

The *goodness* of the achieved solution, i.e., the amount of deviation from the optimum is strongly determined by the type of heuristic employed. Simple (greedy) heuristic algorithms such as hill climbing have the undesirable tendency of getting trapped at local minima of the cost function and are therefore applicable only to a small subset of the problems described above. A stochastic approach such as Simulated Annealing does much better in most applications since it allows temporary cost increases. However, there are certain drawbacks associated with the annealing approach. While the basic algorithm is quite simple and straightforward, an actual implementation requires very careful consideration and tuning of the process parameters to achieve the desired result. In addition, even with an optimally designed algorithm, the computation time required is usually very long compared to most other heuristics.

An algorithm based on the Simulated Evolution heuristic has, therefore, been designed to address these problems. Even though the theoretical proofs of convergence of this new stochastic algorithm are currently being researched, experimentally we have observed promising convergence results. The evolution-based algorithm for standard cell plaement will be described in the next section.

III. Simulated Evolution Cell Placement Algorithm

III.1. Overview of our Proposed Algorithm

The basic idea behind our algorithm is to determine, for each current placement, the *goodness* of each cell in its current location. The *goodness* value is a figure of merit (normalized in the range 0–100) of how well the cell is placed with respect to the cells to which it has connections. The *goodness* is high if most cells it is connected to are clustered together as closely as possible in the present placement. Conversely, the *goodness* is low if those cells are less clustered. The process of *evolution* keeps the cells which are already well placed (having high *goodness* numbers) in their present locations and tries to improve the positions of the others (that have low *goodness*

numbers). At a particular iteration, a random number R in the range 0–100 is generated for each cell, and all cells whose *goodness* values exceed R are labeled as "good cells" and survive in their present locations; the remaining cells are labeled as "bad cells" and do not survive (become extinct) in their present location. The *goodness* number therefore corresponds to the survival chance of a cell in a particular position for a given configuration. The "bad" cells are removed from the placement grid and are allocated to new positions using constructive placement techniques with the objective of clustering cells using a local cost function. The above set of steps is iterated upon. Thus the global placement procedure is based on iterative improvement while within an iteration constructive placement is performed.

The algorithm has two places where probabilistic techniques help prevent the solution from getting stuck at a local minimum. The first one is in the generation of the threshold value to distinguish "good" cells from "bad" cells. The second is in the mutation step. Periodically during the evolution-based iterative improvement process, the system state is *mutated*, i.e., randomly changed. The mutation rate is much less than the evolution rate to ensure convergence. From empirical results we found that a mutation rate of about 0.1 percent of the evaluation rate yields the best results.

III.2. Definitions

In order to facilitate the explanation of the algorithm's operation, we will define the following terms:

Current cell: the cell currently being evaluated;
Current net: the net currently being evaluated;
Neighbor: a cell directly connected to the current cell;
Netset: all nets connected to the current cell.

III.3. Initialization

Our Simulated Evolution based cell placement algorithm is divided into two main stages. A block diagram of the main procedures is shown in Fig. 1. The first routine allows the user to enter certain process parameters if he chooses to override the default settings. Subsequently, the input files containing circuit data and parameters are read. After the aspect ratio of the chip has been determined, the floorplanning algorithm allocates space for the rows in which the standard cells are to be placed. It also takes care of reserving areas for macroblocks and pads which are then preplaced. If the user does not specify fixed locations for them, a routine following the cell placement will optimize their positions in a postprocessing phase. The floorplanning procedure will not only determine the number of rows needed for placement of the standard cells, but also their average and maximal lengths.

The next step is the generation of an initial placement which will serve as a seed for the evolutionary process. Due to the fact that the initial exchange rate is relatively high the algorithm's performance is generally almost independent of the quality of the initial generation. However, if this seed is already a relatively good placement, the overall computation time will decrease. In our program, the seed placement is generated randomly. Optionally, a previously generated placement can be loaded to serve as a seed. This allows an efficient way to evaluating the effects of manually made changes.

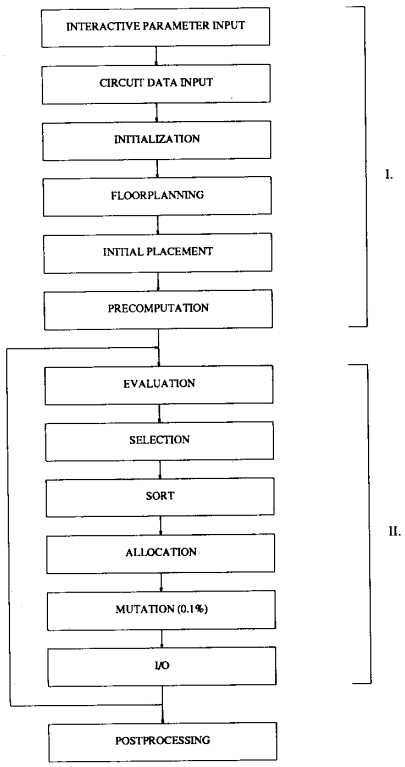


Fig. 1. Algorithm outline.

III.4. Precomputation

The process of evolution keeps the cells which are already well placed in their locations and tries to improve the positions of the others. In order to determine the *goodness* of a cell placed in a particular location, a reference value has to be established. This is done during precomputation.

III.4.1. Wire Length Based Cost Function: For the evaluation cost function which computes the placement value based on the total wire length of a cell's *netset*, lower bounds on all net lengths have to be established in order to allow a meaningful comparison. This is not trivial since there are no restrictions on where the actual pin connection can occur within the cell boundary. The approach taken in ESP is illustrated in Fig. 2. The general method used to calculate a lower bound on the actual net length is to compute half the perimeter of the bounding rectangle. To do this, however, an optimal placement of the cells connected to the *current net* has to be assumed which is the original intent of the cell placement algo-

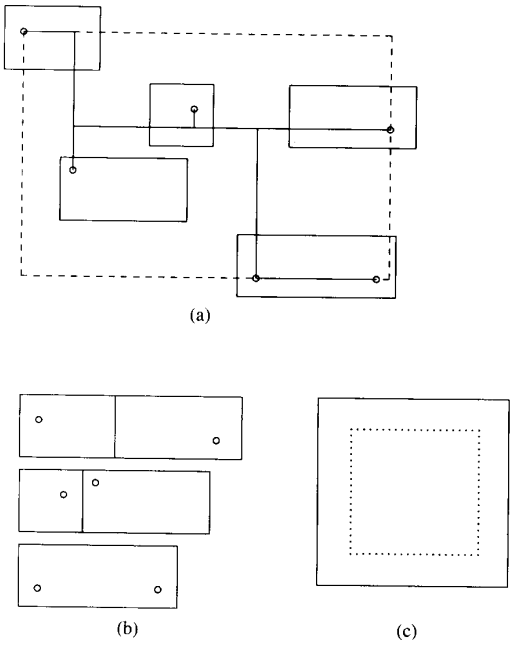


Fig. 2. Precomputation.

rithm! We address this paradoxical situation by focusing on the sub-problem of optimal *relative* placement of cells that belong to the *current net* and ignore (for this precomputation phase) the interactions with cells in other nets. As a first approximation of the optimal placement, all cells of the *current net* are assumed to be placed next to each other without space in between. Next, all cell areas are collapsed to an approximate square region (since a square has the minimal perimeter among all rectangles of the same area). Row spacing is considered and included in the computation. This leads to a minimal perimeter by considering only the cell dimensions. However, the minimal bounding rectangle depends on the relative position of the pins which connect the cells to the *current net*. To estimate the effect on the actual net length, half the height of the cell (equal to the maximal pin displacement in y-direction) is subtracted on each side of the square. The perimeter of the resultant reduced square divided by two serves as the lower bound on the wire length of the *current net*. It has to be stressed, that the computed value is a theoretical result which may not be achieved in practice owing to conflicting requirements regarding cell positions between cells of different nets. This does not have serious adverse effects, however, since the same principle applies to all nets. This cost function computes reference values for nets only.

III.4.2. Iteration-Directed Cost Function: Another method for precomputation, which can also be combined with the one described above, is to use the iterative process itself to compute the reference values for the net lengths. In the beginning, all net lengths are assumed to have an arbitrary large value, e.g., the maximum integer. During each iteration the actual values obtained by the evaluation routine are compared with those reference values. If the actual value is smaller than the reference value, it replaces the now obsolete reference value. We have found that this process converges within a few hundred iterations and leads to acceptable results, thus avoiding the complicated computation described above. The reason for the good convergence is a high probability of each net having a near-minimal net length sometime during the iterative process because of its initially high exchange rates.

The current ESP implementation actually uses a combination of both methods described above. A relaxed version of the first one is used to compute the initial values and the second one is used to update them if necessary. As far as performance is concerned, this combination yields the best results on actual circuits.

III.5. Evaluation

The first step of the iterative loop is the evaluation of the current placement. A placement value for every cell is established. The purpose of computing this measure is to determine which cells are in positions that lead to a minimum total wire length, and which cells contribute unnecessarily to large amounts of additional wire length.

The evaluation procedure, which is done for every net separately, is illustrated in Fig. 3. The following evaluation steps are performed for each net in the design. The wire length of the *current net* is calculated by computing half the perimeter of its bounding box which is the smallest rectangle enclosing all pins belonging to a net.

Then, the ratio of its optimal (precomputed) wiring cost over its current wire length is determined. The result will be called the net's placement value. Subsequently, each cell's *goodness* is computed by taking the average of the placement values of its *netset*. The result is then normalized to a scale from 0 to 100. Averaging over the placement values of all cells, a global *goodness* of the current layout is also computed. Finally, the total wire length of the current placement is calculated.

III.6. Selection

The next step is called the selection phase. Here it is determined whether a cell will retain its current position in the next generation, or if it will be scheduled for new allocation. This is done by comparing its placement value to a random number generated for each cell in the range 0 to 100 percent. If the goodness of the cell is larger than the random number, it will survive in its present position. Otherwise, it is placed in a queue for new allocation. By using this selection process, each cell's chance of survival at its current location is exactly equal to its placement value. The user can, however, alter this default setting by supplying an extra parameter in order to extend or reduce the global survival chances, and thereby control the average number of selected cells per iteration.

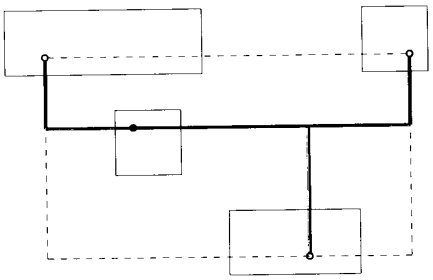


Fig. 3. Wire length based cost function.

III.7. Allocation

The primary operations of the allocation routine are the removal of all selected cells from the placement grid, the re-placement of those cells and a final realignment step to remove empty spaces and overlaps. We have found that the iterative phase can be divided into three major subphases which have unique requirements and demand appropriately suited allocation algorithms. The transition from one stage to the next is dependent on the circuit size and structure.

In order to adapt the allocation routine to different requirements during early, intermediate and final stages of the iterative process, we devised three different allocation procedures. During the early stage, it is important to place a large number of cells since the average global goodness is low causing the number of selected cells q to be high. Due to the fact that the *goodness* of the initial random placement tends to be around 40–50 percent about one half of the total number of cells may be initially allocated for replacement. Due to the complexity of the allocation procedure, this number can be limited for large circuits.

The complexity of the allocation procedure should be low during the early stage. Furthermore, it is sufficient that cells are placed in the general neighborhood of strongly connected cells, however, the exact position may yet be undetermined. It would be costly and unnecessary to compute the optimal position of each cell during every iteration. The *sorted individual best fit* method described below suits these requirements.

After a number of iterations, the gain achieved by the method described above during each cycle will diminish. If no substantial improvement can be achieved during further iterations, the iterative process is considered to have moved to its intermediate stage. At this point, most cells should have reached their approximate positions and long distance moves are rare. Now it is important to reorder the cells within their respective clusters and shape the clusters' boundaries. The allocation function has to operate more accurately, trading off individual gains for global benefits. A few number of cells have to be exchanged during each iteration, saving CPU time for more elaborate computation. We use a *weighted bipartite matching* algorithm during this intermediate phase.

If, after more iterations, even the second method ceases to yield a sufficient gain per iteration, the placement is considered to have almost converged. During this final stage, most cells have reached a close vicinity to their final locations and no long distance movement of cells are necessary. In order to achieve improvement of the current layout, the cost calculations have to be very precise. This requires a large amount of CPU time, therefore only few cells can be exchanged during each iteration. The last allocation method is uses a *branch-and-bound search* algorithm to find the optimal placement of a small set of cells to be newly allocated.

III.7.1. Sorted Individual Best Fit Allocation: All cells which were removed from the placement grid reside in the replacement queue. This queue is sorted, such that the cell with the most connections is placed first. The following steps are performed until all cells have been replaced. Fig. 4 illustrates this process. The algorithm has a time complexity of $O(q^2)$, where q is the number of cells to be replaced which can be relatively large due to the low initial overall *goodness*. Therefore, the program limits this number to a maximum of about 100 cells.

The first cell is removed from the allocation queue and tentatively placed at all empty locations. During this process, a simplified evaluation routine is invoked, which determines the *goodness* of the *current cell* at each trial location. This evaluation routine can, of course, only compute wire lengths involving already placed cells. The best placement value for the cell is retained and determines its new location. The *current cell* is placed in the new position. Subsequently, the next cell in the queue is processed, using the remaining empty locations.

III.7.2. Weighted Bipartite Matching Allocation: The basic principle of this method is shown in Fig. 5. Before the actual replacement starts, the *goodness* of each cell in the queue is evaluated in every possible location. This yields a table of size q^2. The matching algorithm now tries to assign each cell to a location, such that the overall wire length is minimized. This can be done optimally in $O(q^3)$ time complexity [25]. This is an increased complexity compared with the first method, however, the number of cells q to be replaced is now much smaller, about 20–30 cells. The computation can only be as accurate as the preceding evaluation. Therefore, wires connecting unplaced cells are not considered. The overall accuracy is, however, much better than in the first method. Sorting of cells is not explicitly needed, but the procedure benefits if the order of cells in the minimal cost assignment path coincides with the ordering of the allocation queue. We found empirically, that ordering the cells such that the ones with the smallest *goodness* values (from the previous cycle) are processed first improves the algorithm's performance.

III.7.3. Branch-and-Bound Search Allocation: The branch and bound method of exhaustive search works by continually trying to extend a partial solution. If an extension of the current solution is not possible, a backtrack is performed to a shorter partial solution. It is based on the assumption that each solution has a cost associated with it which is well defined for partial solutions, and has

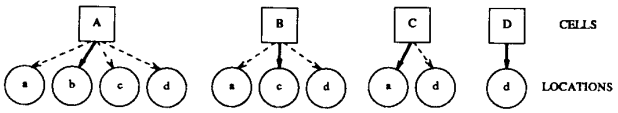


Fig. 4. Sorted individual best fit allocation.

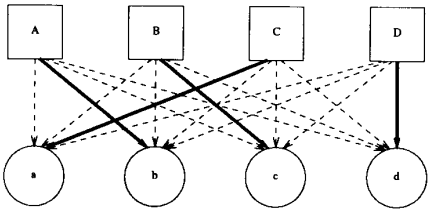


Fig. 5. Weighted bipartite matching allocation.

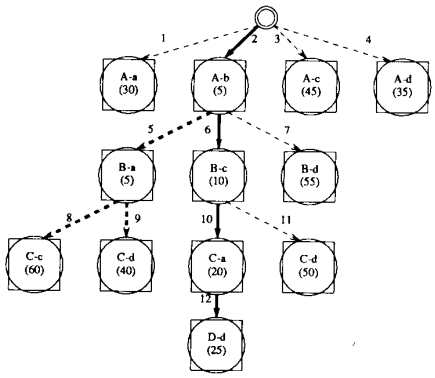
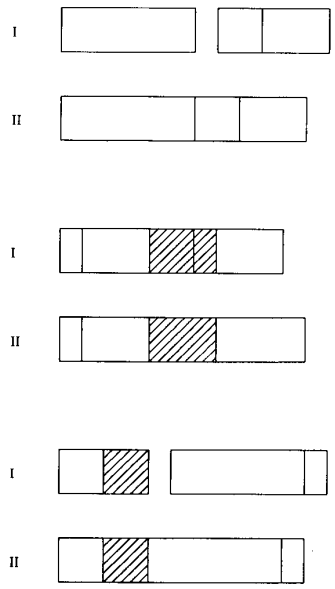


Fig. 6. Branch-and-bound search allocation.



I: Module positions after allocation, before alignment

II: Module positions after alignment

Fig. 7. Operation of the alignment precedure.

the property that for all partial solutions and their extensions, the cost of the extension is lower bounded by the cost of the partial solution [26]. The success of this method is based on an accurate calculation of the cost function that has the above mentioned property of bounding. The search tree of an optimal allocation of a small number of cells q is shown in Fig. 6. Each node in the tree corresponds to an assignment of a cell to a location. The basic idea is to calculate lower bounds on the wire length incurred at each tree node (this cost is shown in parenthesis). This is done by calculating the wire lengths of the current cell's *netset*. Unplaced cells are not considered. This does not affect the accuracy of the procedure since only a lower bound needs to be computed. During the placement of the last cell in the replacement queue, all nets are accurately evaluated, therefore the search is guaranteed to find the optimal placement.

The algorithm always expands the tree node having the least cost so far and evaluates its children. This evaluation can be done in two parts, the static and the dynamic part. The static subset contains all nets which are connected to only one unplaced cell. The wire lengths of this set can be precomputed by tentatively placing every unplaced cell in all possible locations. The resulting costs are stored in a table of size $O(q^2)$, as in the previous algorithm. The dynamic subset contains all nets connected to more than one unplaced cell. The intersection of the dynamic part and the *netset* of the *current cell* has to be recomputed for each tree node.

The overall complexity of this method is clearly exponential, however, sorting the replacement queue can force early bounding and reduce the required CPU time. Therefore, the cells potentially incurring the highest cost due to their number of connections, are placed in the front of the replacement queue. In addition, the number of cells q to be replaced is now very small, usually less than 10 cells.

III.7.4. Cell Realignment after Allocation: Since the cell which is to be placed and the empty slot are generally of different width, special consideration has to be given to the problem of possible overlaps and unused spaces. This is important because the subsequent realignment of the cells can cause the expected improvement in the total wire length to be diminished or even negated. So far, we have found that realignment after each allocation phase does not seriously disturb the convergence of the algorithm. Removing the overlaps and unused spaces between the cells simplifies the program and yields legal placements at any stage of the iteration.

The current implementation employs a routine which cleans up all overlaps and unused spaces between cells after the allocation step is completed. The operation of the realignment routine is shown in Fig. 7. Ongoing research is done on studying the effects of allowing temporary overlaps and empty spaces. We are also trying to estimate the additional cost of potential realignment during the allocation phase.

III.8. Mutation

The first step of this procedure is to determine if a mutation will occur in the current iteration. The probability of a mutation is 0.1 times the current cell exchange rate. If a mutation is to occur the procedure randomly selects two cells from the placement grid and exchanges them without regard to the effects on the placement values. This step is repeated according to user-supplied parameters. In case of unequal cells, a collision-resolving routine is called to clean up the placement grid.

III.9. I/O and Postcomputations

The remaining part of the main loop is dedicated to I/O operations such as informing the user of the current wire length, updating the performance statistics, and in case an improvement has been achieved, saving the current placement. If the user wishes to change certain parameters, the computation loop can be interrupted at this point to invoke the interative parameter input routine. The loop termination condition is checked, and depending on its result, the program branches to the loop start or to the postcomputation routine.

The postcomputation procedure creates certain output files containing the placement information and performance statistics. It optionally invokes routines which optimize pad and macroblock placement.

III.10. ESP compared to Simulated Annealing and Genetic Placement

Both our ESP and SA are algorithms using iterative improvement strategies. They try to avoid local minima by encorporating random elements into the guiding heuristics. However, there are several major differences between the approahces taken by ESP and SA as we will outline in the following.

Most SA algorithms use *random* choices of cells to be moved to alternate locations. Typically in one move, one cell is displaced to a new location, or two cells are exchanged. ESP allows the simultaneous removal of a large number of *badly placed* cells from their current locations using a stochastic method and places them in new locations.

During each iteration, SA performs tentative exchanges and accepts or rejects them based on a Boltzmann distribution function. ESP, however, tries to construct a variety of partial solutions during each iteration and combines them to a current optimum. In this respect, ESP has some similarities to the constructive placement algorithms mentioned earlier.

The SA algorithm requires an explicit parameter (called temperature) to guide the iterative process, while ESP implicitly analyzes the current placement *goodness* and uses it to guide its operation.

As mentioned before, another evolution-based approach for standard cell placement, called Genetic Placement (GP) has recently been proposed [20]. The general strategy of using evolution from one generation to the next is quite similar to our approach. There are, however, significant differences in the way a new generation is constructed as will be shown in the following.

The GP technique maintains a set of solutions during each iteration, called parents. It then constructs another set of solutions for the next generation by merging the placements of two parents to obtain one child. ESP, however, generates only one child from one parent during each generation, thereby eliminating the extra CPU time and memory needed to maintain a set of solutions.

Furthermore, the methods of selecting cells to be replaced are totally different. While the GP algorithm selects a random set from the parents, ESP will determine the *goodness* of each cell individually to determine its survival at its present location. Due to these stronger convergence properties, it performs much less total iterations than GP.

In conclusion, we are convinced that the application of evolution to engineering problems is a reasonable approach which leads to uncomplicated and easily adaptable algorithms. ESP may serve as an example for a successful implementation. Its main advantage is a comparatively fast execution time while achieving results of high quality.

IV. Placement Results

We compared our algorithm to the Simulated Annealing based placement programs TimberWolf 3.2 [16] and TimberWolf 4.1 [17] in placement-only mode using their recommended annealing schedules to produce near-optimal results. However, on several circuits, we were still able to obtain some improved results. The CPU time needed by our algorithm was in all cases at least one order of magnitude smaller that the computation time required by TimberWolf 3.2 and in most cases significantly faster than TimberWolf 4.1. It should be noted that TimberWolf 4.1 goes beyond the basic Simulated Annealing principle by adding heuristics based on statistical data generated by numerous placements.

All computations were carried out on a SUN 3/50 desktop workstation with MC68020 CPU and MC68881 floating-point-coprocessor. Both ESP and TimberWolf were compiled using the floating point option and code optimization and were executed under the BSD UNIX 4.2 operating system. It should be noted that TimberWolf was written in the C programming language using a lot of sophisticated programming techniques such as hashing into bins, etc., to maximize the performance. ESP is written in PASCAL (using 5000 lines of code) for ease of programming and does not use any sophisticated programming techniques since our intent was to demonstrate the concept and not develop a mature product. We believe that we can speed up our ESP program by an additional factor of 3–4 by implementing it in C and including sophisticated programming features.

The placement results for several different circuits are compiled in Table I. Most circuits were obtained from industry while some (e.g., the 1000 cell circuit) were gen-

TABLE I
ESP RESULTS

circuit		ESP		TimberWolf 3.2		TimberWolf 4.1	
# rows	# cells	time [s]	result	time [s]	result	time [s]	result
8	60	141	17021	3082	19528	464	18007
13	183	1568	73321	9669	73566	1284	62965
16	286	7762	151378	20116	139777	2775	108625
32	300	2412	188096	62712	195350	7452	179408
31	1000	22317	304380	927801	310120	32133	333040

TABLE II
COMPARISON OF ALLOCATION STRATEGIES

Placement Status	Allocation Algorithm	Performance Index *
Initial	Best fit	**730**
	Matching	633
	BB. Search	118
Intermediate	Best fit	206
	Matching	**298**
	BB. Search	102
Final	Best fit	14
	Matching	3
	BB. Search	**21**

* Average wire length reduction per CPU second

erated artificially to simulate circuits with high connectivity. Each row in the table shows values for a different circuit. For each circuit, the CPU time required and the result obtained is listed. All placement results reflect the estimated total wire length of the best placement using the bounding rectangle criterion (half the perimeter equals the estimated wire length).

As the data in Table I shows, the placement values generated by ESP are comparable to those generated by the TimberWolf programs. In some cases TimberWolf obtains a better result whereas on other circuits the placement generated by ESP is superior. We are currently working on an improved version of the ESP program; the prime objective being to consistently obtain results of highest quality.

The data in Table II shows the performance of the different allocation methods during different stages of convergence. The performance index shown in the last column was measured using a standard 300 cell circuit. The program was given an initial placement as stated in the first column and run for a certain constant amount of time. Then, the improvement of the total wire length was determined and divided by the CPU time used.

From these measurements, it can be deducted that the *sorted individual best fit allocation* method performs best during the initial stage. During the intermediate stage, the *weighted bipartite matching allocation* method shows superior preformance. Finally, the *branch and bound search allocation* method gives the best result near convergence.

V. CONCURRENT ESP

V.1. Motivation

As we have shown in the preceding section, the Simulated Evolution algorithm provides good placement results using relatively little CPU time. However, the placement of circuits containing several thousand cells will still take many hours to complete. Therefore, we studied possibilities of achieving an even greater speedup. One possibility is to use a faster computer such as a mainframe instead of a workstation. An alternative is to use a network of workstations as a distributed processing environment to provide an efficient and cost-effective way to compute large placements. Due to its structure, the placement problem seemed a good target for such an implementation, as will be outlined subsequently. Our initial results on such a scheme has been reported at the ICCAD conference [27]. In the remainder of this paper, we will discuss the distributed algorithm and its implementation in detail. Similar approaches have been pursued by other researchers for parallelizing Simulated Annealing based cell placement algorithm on multiprocessor systems [28]–[31].

V.2. Workload Partitioning

When redesigning an algorithm in order to use it on a system with distributed resources, the main task is to partition the workload most efficiently. This divide-and-conquer approach is guided by the following aspects and considerations:

1) The total amount of CPU time on all machines combined should not significantly exceed the amount of CPU time needed on one machine. To achieve this goal, the cost of partitioning, scheduling, joining, and communication has to be kept low, compared to the overall computation time.
2) The amount of real time needed for execution on parallel machines should approximately equal the amount of real time needed on one machine divided by the number of machines used, or in other terms, the computation efficiency should be as close to unity as possible.
3) For synchronization purposes, the workload, expressed by the amount of real time required on each machine, should be equally distributed, not only considering the total amount, but also for each iteration. Significant deviations in the workload will slow down the process, since each iteration can begin only after all previous data has been successfully collected.
4) The layout quality product by the distributed algorithm should equal the performance of the single machine version.

The partitioning in the concurrent algorithm is done by assigning a certain subset of the cells to each machine as shown in Fig. 8 for the case of three machines. The different shadings show the allocation of rows to the differ-

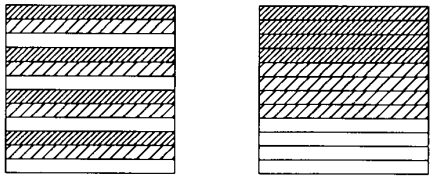


Fig. 8. Row partitioning for three machines.

ent hosts. The left pattern shows the distribution for odd-numbered iterations and the right one the partitioning for even-numbered cycles. It should be noted that only two different patterns are needed for any (small) number of processors. The partitioning is done row-wise because this greatly simplifies the adaptation of the algorithm to the parallel environment and avoids adverse effects created by unequal cell lengths.

The pattern for each host alternates every iteration to ensure that each cell can move to any position on the grid in at most two steps, as long as the number of rows per machine is sufficiently large. In that case, boundary effects which lead to an increased number of necessary moves can be neglected.

We will now describe how the requirements 1.–4. mentioned above are taken care of in the distributed version of the algorithm.

The partition of the workload in terms of rows in the placement grid satisfies the requirements of an approximately equal distribution. The computational overhead due to the distributed structure is kept low. This is mostly due to the fact that a complete iteration is performed by each machine without the need for any communication during the process. The reason for partitioning the tasks at the block-level is the relatively high cost of initiating a communication on the bus.

The total number of real time needed for the program to execute is dependent on many factors, mainly the workload on each machine and the network traffic. Due to the necessary synchronization after each cycle, the total amount of real time needed will equal the sum of real times required by the slowest machines during each iteration. Since an equal amount of cells is assigned to each host, the computation times are not significantly different. Little variations, however, are not avoidable due to the random processes involved. Irregularities tend to cancel out with an increasing number of cells per machine.

The final placement quality depends mainly on the general structure of the algorithm, and less on the actual conditions during each single iteration. The main strategy, however, has remained the same so that comparable results of the sequential and the distributed version can be expected.

VI. Implementation of Concurrent ESP

The main program structure consists of a master program and one or more servers which get their instructions remotely from the master during initialization. The master partitions the workload and schedules the tasks to be performed on each machine. The block diagram of the concurrent algorithm is shown in Fig. 9.

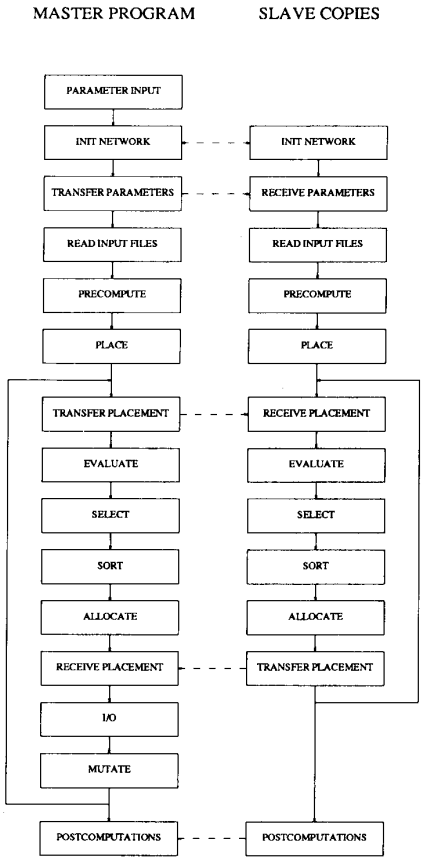


Fig. 9. Outline of the distributed algorithm.

It should be noted that only the main iterative loop is computed in a distributed manner with true parallelism and workload distribution. The precomputations are performed by all machines concurrently, but in the same sequential way as in the uniprocessor case, since the amount of CPU time required is very small compared to the CPU time needed during the iterative process (usually less than 1 percent). Therefore, the overall performance of the program is not degraded by this mode of operation. The communication is, due to the network implementation, packet-switched and based on the client/server model.

After performing the interactive parameter input, the master initializes one or more servers, as specified by the user. After successful initialization of the network, all program copies perform the same precomputations. The corresponding blocks in Fig. 9 are *read input files*, *precompute*, and *place*. The workload partitioning is done by a special routine which is executed during the precomputation phase.

At the beginning of each iteration, the master program distributes the current placement information (i.e., the locations of all cells on the placement grid) to all its servers. Next, each process is assigned its subset of cells, accord-

ing to the currently active partition. Then, each copy including the master itself, performs a complete cycle consisting of *evaluate*, *select*, *sort*, and *allocate*.

Having completed the allocation routine, each process sends its results back to the master which combines them and performes the *I/O* and *mutate* tasks. The master determines if a new iteration is to be started or the loop is to be exited. In the latter case, postcomputations will be performed and all copies terminate execution.

VII. Performance Evaluation of Concurrent ESP

The computations were carried out on several SUN 3/50 desktop workstations connected to a file server by a common ETHERNET bus with a nominal data transfer rate of 10 Mbit/s. The workstations are equipped with an MC68020 CPU and an MC68881 floating-point coprocessor. The program, written in PASCAL, was compiled using the code optimization and floating-point options and executed under the BSD UNIX 4.2 operating system.

To evaluate the performance of the distributed algorithm, five similar circuits were used. The circuits contain the same type of cells and have the same interconnection density. They differ, however, in the number of cells. Similar circuits were used in order to compare the performance as a function of the number of cells per circuit only while minimizing other dependencies. For each circuit, the number of rows was selected automatically by ESP such that the x- and y-dimensions of the chip were approximately equal (to minimize the total wire length).

For each circuit, the placement program was executed for 1000 iterations on one to four processors. The CPU time and real time requirements were measured along with the final placement value (minimal total wire length). A summary of the results is given in Table III.

From the data obtained during the experiments, speedup and efficiency factors as well as degradation of the result were calculated for each case. The speedup diagram for each circuit is shown in Fig. 10, the degradation of the final results versus the number of machines used is shown in Fig. 11.

$$\textit{Speedup:} \quad S = T_1/T_p$$

$$\textit{Efficiency:} \quad E = S/p$$

$$\textit{Degradation:} \quad D = R_1/R_p$$

T_1 time required by a single processor,
T_p time required by p processors,
R_1 result achieved with a single processor,
R_p result achieved with p processors,
p number of processors.

A closer look at the results reveals the efficiency exceeds the maximum value of unity in certain instances. This behavior can be explained by considering the derivation of the values and the nature of the algorithm. Basically, three factors are responsible for that effect:

1) The actual amount of computation depends on the number of cells exchanged during each iteration, which is in turn a function of the current *goodness*. Due to the statistical nature of the algorithm, the amount of computation differs to a certain extent each time the program is run, as does the finally achieved near-optimal result.
2) The efficiency calculation does not take into account the degradation in the quality of the final result. This degradation is due to the increased granularity of the partitioned problem compared to the sequential version. Therefore, the additional amount of computation needed in a postprocessing step to achieve the

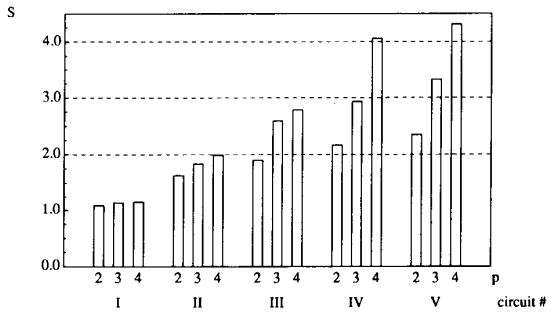


Fig. 10. Speedup versus number of machines.

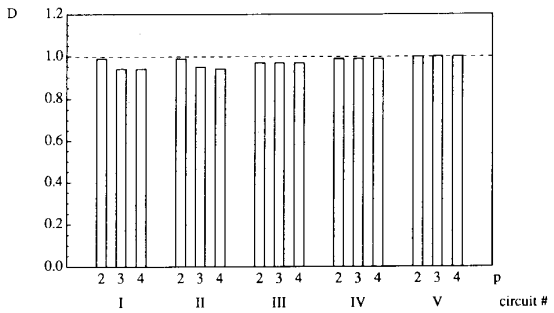


Fig. 11. Result degradation versus number of machines.

TABLE III
Concurrent ESP Results

circuit	# cells / # rows		# processors 1	2	3	4
I	200	real time	575	527	504	499
		cpu time	573	405	354	337
	10	result	30500	30900	32510	32330
II	400	real time	1987	1220	1085	998
		cpu time	1974	973	714	672
	14	result	83540	84100	87600	88800
III	600	real time	4461	3252	1724	1605
		cpu time	4444	1904	1236	1080
	17	result	153080	157700	158540	158080
IV	800	real time	9015	4166	3070	2215
		cpu time	8979	3145	2415	1635
	20	result	229430	231290	231140	231300
V	1000	real time	13371	5697	4010	3100
		cpu time	13325	4847	3090	2190
	22	result	327580	327590	327600	327650

same result as in the uniprocessor case should be considered. Due to the heuristic nature of the algorithm, however, no exact value for the additional CPU time required can be given.

3) It may be possible to reduce the amount of computation performed in the uniprocessor case by deliberately reducing the search space during the allocation process, as forced in the distributed implementation. We are currently investigating this possibility and its implications.

As can be deduced from the graphs, if each processor is assigned at least 250–300 cells, the parallel processing efficiency reaches 100 percent without any serious degradation of the final result. This proves the effectiveness of the workload partitioning.

VIII. Conclusions and Future Work

In this paper, we have proposed a new heuristic called Simulated Evolution for standard cell placement that is based on an analogy between the natural selection process in biological environments and the method of solving engineering problems by iterative improvements.

The ESP program applies the Simulated Evolution method to the standard cell placement problem. The combination of evolution and mutation has been experimentally observed to give good results for realistic circuits. Various precomputation, evaluation, and allocation strategies have been experimented with and were reported in the paper. The ESP placement package has been shown to achieve results comparable to SA algorithms, using less the CPU time for results of comparable quality. We expect additional speedups in future versions of ESP due to an improved algorithm and the use of the C language which will allow a more efficient coding.

A concurrent implementation of the ESP placement algorithm has also been presented. The design uses desktop workstations connected by a local area network as processing elements. With regard to these hardware constraints, a general model for the distributed algorithm has been developed. The concurrent algorithm has been implemented in a program package and tested on a variety of circuits. Its performance has been measured and the results presented. We have shown that the total communication overhead is low and that an efficiency close to unity can be achieved, given that each processor is assigned a sufficiently large subset of cells. The concurrent algorithm allows the placement of large circuits (several thousand cells) on desktop workstations within acceptable computation times, thereby eliminating the necessity of a large mainframe computer.

Our paper has shown the application of an evolution-based heuristic for solving problems which are not easily solved in a closed form by using the cell placement application as an example. We believe that evolution-based heuristics have a variety of features which make them useful in almost all areas of engineering. Our future research will focus on the improvement of the ESP placement program and the development of an evaluation-based global and detailed routing program. This will allow chip designers to use ESP as an integrated physical chip design package. We are also investigating the theoretical aspects of convergence properties of evolution-based heuristics for combinatorial optimization problems using the concept of Markov chains.

References

[1] S. Sahni and A. Bhatt, "The complexity of design automation problems," in *Proc. 17th Design Automation Conf.*, pp. 402–411, June 1980.

[2] W. E. Donath, "Complexity theory and design automation," in *Proc. 17th Design Automation Conf.*, pp. 412–419, June 1980.

[3] B. T. Preas and P. G. Karger, "Automatic placement: A review of current techniques," in *Proc. 23rd Design Automation Conf.*, pp. 622–629, June 1986.

[4] M. Hanan and J. M. Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems: Theory and Techniques*, M. A. Breuer, Ed. Englewood, NJ: Prentice-Hall, pp. 213–282, 1972.

[5] M. R. Hartoog, "Analysis of placement procedures for VLSI standard cell layout," in *Proc. 23rd Design Automation Conf.*, pp. 314–319, June 1986.

[6] M. A. Breuer, "Min-cut placement," *J. Design Automation and Fault Tolerant Computing*, vol. 1, pp. 343–382, Oct. 1977.

[7] B. W. Kernighan and S. Lin, "An efficient heuristic for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, Feb. 1970.

[8] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 92–98, Jan. 1985.

[9] M. Hanan and J. M. Kurtzberg, "A review of the placement and the quadratic assignment problem," in *SIAM Rev.*, pp. 324–342, Apr. 1972.

[10] J. P. Blanks, "Near-optimal placement using a quadratic objective function," in *Proc. 21st Design Automation Conf.*, pp. 602–615, June 1985.

[11] D. G. Schweikert, "A 2-dimensional placement algorithm for the layout of electrical circuits," in *Proc. 13th Design Automation Conf.*, pp. 408–416, June 1976.

[12] M. Hanan, P. K. Wolff Sr., and B. J. Agule, "Some experimental results on placement techniques," in *Proc. 13th Design Automation Conf.*, pp. 214–224, June 1976.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[14] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proc. Int. Conf. Computer-Aided Design.*, pp. 381–384, Nov. 1986.

[15] C. Sechen and A. S. Vincentelli, "The TimberWolf placement and routing package," in *Proc. Custom Integrated Circuits Conf.*, pp. 522–527, May 1984.

[16] C. Sechen and A. S. Vincentelli, "TimberWolf 3.2: A new standard cell placement and global routing package," in *Proc. 23rd Design Automation Conf.*, pp. 432–439, June 1986.

[17] C. Sechen and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-based Placement," *Proc. Int. Conf. Computer-Aided Design*, pp. 478–481, Nov. 1987.

[18] L. K. Grover, "A new simulated annealing algorithm for standard cell placement," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 378–380, Nov. 1986.

[19] R. M. Kling and P. Banerjee, "ESP: A new standard cell placement package using simulated evolution," in *Proc. 24th Design Automation Conf.*, Miami Beach, FL, pp. 60–66, June 1987.

[20] J. P. Cohoon and W. D. Paris, "Genetic placement," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 956–964, Nov. 1987.

[21] S. Chang, "The Generation of minimal trees with Steiner topology," *J. Assoc. Comp. Mach.*, vol. 19, pp. 699–711, Oct. 1972.

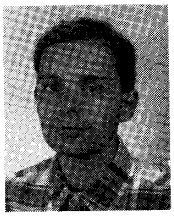
[22] R. Otten and L. P. P. Ginnekan, "Floor plan design using simulated annealing," in *Proc. Int. Conf. Computer-Aided Design*, vol. CAD-3, pp. 96–98, Nov. 1984.

[23] G. Persky, "PRO—An automatic string placement program for polycell-layout," in *Proc. 13th Design Automation Conf.*, pp. 417–424, June 1976.

[24] B. Dunham *et al.*, "Design by natural selection," in *Synthese*. Dordrecht-Holland: Reidel, pp. 254–259, 1963.

[25] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
[26] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Englewoods-Cliffs, NJ: Prentice-Hall, 1977.
[27] R. M. Kling and P. Banerjee, "Concurrent ESP: A placement algorithm for execution on distributed processors," in *Proc. Int. Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 354-357, Nov. 1987.
[28] S. A. Kravitz and R. A. Rutenbar, "Placement by simulated annealing on a multiprocessor," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 534-549, June 1987.
[29] M. Jones and P. Banerjee, "Performance of a parallel algorithm for standard cell placement on the Intel hypercube," in *Proc. 24th Design Automation Conf.*, Miami Beach, FL, pp. 807-813, June 1987.
[30] A. Casotto, F. Romeo, and A. S. Vincentelli, "A parallel simulated annealing algorithm for the placement of macro-cells," *IEEE Trans. Computer-Aided Design*, pp. 838-847, Sep. 1987.
[31] J. S. Rose, D. R. Blythe, W. M. Snelgrove, and Z. G. Vranesic, "Parallel standard cell placement algorithms with quality equivalent to simulated annealing," *IEEE Trans. Computer-Aided Design*, pp. 387-396, Mar. 1988.

*

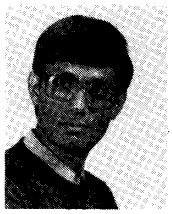


Ralph M. King received the B.S. degree in electrical engineering from the University of Hannover, West Germany in 1984, and the M.S. degree in electrical engineering from the University of Illinois at Urbana-Champaign in 1987, where he is working toward the Ph.D. degree in electrical engineering.

He was a Research Assistant at the Institute for Semiconductor Physics at the University of Hannover during 1985. Since 1986, he has been a Research assistant with the Computer Systems Group at the University of Illinois. In addition, he has worked at Siemens AG (1982), at IBM Germany (1985), and the IBM T.J.Watson Research Center during the summer of 1988. His research interests include computer aided design in integrated and design automation, as well as computer architecture and parallel processing.

Mr. Kling was awarded a scholarship from the West German government for outstanding academic achievements in 1983. He also received a Fulbright Scholarship for studies in the United States in 1985.

*



Prithviraj Banerjee (S'82-M'84) received the B.Tech. degree in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1982 and 1984, respectively.

He is currently Assistant Professor of Electrical and Computer Engineering and the Coordinate Science Laboratory at the University of Illinois at Urbana-Champaign. His research interests are in Fault-Tolerance in Multiprocessor Systems, and Parallel Algorithms for VLSI Design Automation. He is also consultant to Westinghouse Corporation, the Research Triangle Institute, and the Jet Propulsion Laboratory.

Dr. Banerjee received of the President of India Gold Medal from the Indian Institute of Technology, Kharagpur, in 1981, the IBM Young Faculty Development Award in 1986, and the National Science Foundation's President Young Investigators' Award in 1987. He has served on the Program and Organizing Committees of the Fault Tolerant Computing Symposia, and is the General Chairman of the International Workshop on Fault Tolerance in Multiprocessors.