Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing

JONATHAN S. ROSE, MEMBER, IEEE, W. MARTIN SNELGROVE, MEMBER, IEEE, AND ZVONKO G. VRANESIC, SENIOR MEMBER, IEEE

Abstract - Parallel algorithms with quality equivalent to the simulated annealing placement algorithm for standard cells [23l are presented. The first, called *heuristic spaning,* creates parallelism by simultaneously investigating different areas of the plausible combinatorial search space. It is used to replace the high temperature portion of simulated annealing. The low temperature portion of Simulated Annealing is sped up by a technique called *section annealing,* in which placement is geographically divided and the pieces are assigned to separate processors. Each processor generates Simulated Annealing-style moves for the cells in its area, and communicates the moves to other processors as necessary. Heuristic spanning and section annealing are shown, experimentally, to converge to the same final cost function as regular simulated annealing. These approaches achieve significant speed-up over uniprocessor simulated annealing, giving high quality VLSI placement of standard cells in a short period of time.

I. INTRODUCTION

AS DESIGNERS have come to rely on automatic layout tools, it has become necessary that those tools have the ability to do a good job minimizing the final area and other performance-critical factors. Recent work on automatic placement for standard cells [23],]24] has shown that a simulated annealing [151 placement algorithm can achieve higher quality results (lower final area) than more conventional algorithms. The better quality comes at the price of much longer computation time, on the order of weeks on a VAX 11/780 machine [24].

This paper presents techniques for achieving the same final quality as Simulated Annealing, suitable for implementation on an MIMD (Multiple Instruction Stream Multiple Data Stream) multiprocessor and resulting in much faster run times [191, [20]. Two different approaches are taken, one to *replace* the high temperature portion of simulated annealing, and the other to speed UP the low temperature portion. The high temperature approach, *heuristic spanning,* achieves parallelism by having different processors investigate plausible but independent areas of the combinatorial search space.

The low temperature approach, *section annealing,* divides the interim placement into geographic areas and assigns these areas and the cells contained in them to separate processors. Each processor generates simulated annealing-style moves for its assigned cells, communicating accepted moves to other processors when necessary. This approach has been implemented on a five-processor prototype, and expected results are given for ten processors.

There has been a great deal of interest in speeding up the simulated annealing placement algorithm. Kravitz and Rutenbar [161, [171, [22] have provided two approaches to the problem: one that uses pipelining and direct parallelism to speed up the original simulated annealing algorithm, achieving a speed-up of about two using three processors. They also attempt a *parallel moves* strategy on a shared-memory machine multiprocessor, gaining a speed-up of about three using four processors. Banerjee and Jones [1] discuss using a distributed memory Hypercube architecture for the standard cell placement problem. Casotto *et al.* [4] worked on speeding up simulated annealing for placement of macrocells, and have achieved a speed-up of six using eight processors.

Our contribution is in several areas: replacing the high temperature portion of simulated annealing with heuristic spanning is a whole new way of approaching the problem and of obtaining parallelism. The schcmc succeeds by making intelligent use of a limited number of processors.

The idea of generating and evaluating simulated annealing moves in parallel is common to [I]. [4], [16], [20]. We are able to concentrate on the low tcmpcrature phase for this approach, since Heuristic Spanning adequately replaces the high temperature phase. Our section annealing approach uses distributed local memories rather than the shared memory used in [161, and thus does not suffer from the central bottleneck of a shared memory. As opposed to [161, we allow simultaneous move acceptance to occur, and present some new results on the effect of this error on the convergence of the section annealing process. The implementation of section annealing has established the existence and solution of several problems that were not foreseen in [I]. We also introduce a technique to reduce the synchronization cost between processors, by taking note of the fact that every processor does not need to know about every move that is accepted by other processors. Our results are based on experiments with large, industrial circuits, ranging in size from 446 cells to 1795 cells.

Manuscript received January 5, 1987; revised June 16, 1987. This work was supported by Bell Northern Research. Ltd., and by the NSERC CRD Grant 8438. The review of this paper was arranged by Editor A. J. Strojwas.

J. S. Rose is with the Computer Systems Laboratory. Stanford University, Stanford, CA 94305

W. M, Snelgrove and Z. Ci. Vranesic are with the Department of Elcctrical Engineering, University of Toronto. Ont., Canada.

IEEE Log Number 87183Y8.

The experimental work in this paper was performed on a multiprocessor consisting of six National Semiconductor 32016 processors, each with 1 Mbyte of local memory. They communicate through a MULTIBUS backplane, also making use of 1 Mbyte of global memory. The operating system of the multiprocessor is master/slave TUNIS [2], a UNIX-like multiprocessor operating system research project at the University of Toronto. The system was programmed with the Concurrent Euclid language [6], a descendant of Pascal that uses Hoare's monitors [111 for interprocessor synchronization.

This paper is organized as follows. Section II discusses the general task of parallelizing an application, and then the specifics of doing so for simulated annealing. Section III presents heuristic spanning, an approach for replacing the high-temperature portion of Simulated Annealing. Section IV presents Section Annealing, a technique for speeding up low-temperature Simulated Annealing.

II. Parallelism Considerations

In exploring ways to speed up an application through parallelism, the choices of multiprocessor architecture and programming approach are crucial. The multiprocessor architecture must be flexible enough so that both the algorithm and its data structures can be changed easily, since any production application software must be continuously adjusted and corrected throughout its useful life. Previous special-purpose hardware for placement [131, [25] used algorithms that were fixed in hardware, and suffered from the inability to change, or tune their algorithms. In addition, a given architecture is more economic if it is general enough to be used in a range of applications. For these reasons, all approaches discussed in this paper assume a general purpose MIMD multiprocessor.

2. I, *Parallelization of Algorithms*

In attempting to speed up an application that focuses on a specific uniprocessor algorithm using a parallel processor there are two possible approaches:

1) Speed up the serial code for the application, by finding places where it can be pipelined or directly executed in parallel, always maintaining the exact behavior of the algorithm.
2) Try to reproduce the behavior and results of the existing algorithm, but use a different, more parallel approach.

The first approach was used by Kravitz and Rutenbar in their *StaticFunction* implementation [161. It can achieve some speed-up but, as they found, it restricts the parallel programmer's freedom greatly. In general, the total speed-up is not likely to be more than 4 or 5 unless the algorithm has obvious independent parallelism.

The second approach allows much more freedom. The algorithm can be adjusted slightly or changed completely to gain parallelism. It is important that an implementation of the original algorithm is available so the final results of the new approach can be measured against that standard. In this work, we apply the second approach, with the aim of obtaining a greater degree of parallelism.

2.2. *Uniprocessor Simulated Annealing*

Our work is based on that of Sechen and Sangiovanni-Vincentelli [23]. We have implemented a version of their standard cell placement algorithm, which is called SALTOR for Simulated Annealing Layout at Toronto [191. It begins using a random layout, and a high temperature (T) where the acceptance ratio is over 60 percent. Continuous placement perturbations called *moves* are generated, and the change in cost function that each move would cause (AC) is calculated. The move is accepted if $AC \leq 0$ (i.e., it improves the cost function) or with probability $e^{-\Delta C/T}$ if $AC > 0$. The temperature is decreased by a constant factor (we typically used 0.85) after a constant number of moves per cell were attempted (typically 100). There are two kinds of moves: the displacement of single cells over a random distance, and the exchange of two randomly chosen cells. Moves are *range-limited:* a range window, which decreases logarithmically in size with the temperature, gives the maximum displacement of one cell, and the maximum distance over which two cells can be exchanged. We did not implement the cell orientation move type or the low-temperature intra-row exchange step of [23], because the basic properties of Simulated Annealing are captured by the displacement and exchange moves.

2.3. *Relevant Characteristics of Simulated Annealing*

The Simulated Annealing algorithm exhibits markedly different characteristics at different temperatures. The early stages, at high temperatures, are characterized by high acceptance ratios, and the moves involve large distances across the entire circuit. The later stages, at low temperatures, exhibit low acceptance ratios and the moves cover small distances with respect to circuit size [191.

The acceptance ratio and average size of move made are the key factors in parallelizing Simulated Annealing. The acceptance ratio dictates how often the data structures that contain cell positions and wire lengths must be changed. If the ratio is high then any parallel approach that needs to access a single database containing that information will suffer from severe bottlenecks. The average size of move dictates, to some extent, the *locality* of the work in the database, and thus will affect multiprocessor implementations with distributed caches or local memories. Also, since we have decided to reproduce the *behavior* (and not the identical algorithm) of Simulated Annealing, we note that the behavior of the algorithm is entirely different for high and low acceptance ratios. For these reasons, different approaches to parallelizing Simulated Annealing must be used for the high and low temperature ranges.

III. High Temperature: Heuristic Spanning

In the *regular* Simulated Annealing algorithm at high temperatures, the general area of the search space being

investigated changes rapidly due to the high acceptance ratio and large scale of moves (by regular Simulated Annealing we refer to the uniprocessor method over the full temperature range, as described in Section 11-2.2). The result of the high temperature phase is a coarse placement that assigns each cell to a general area.

An alternative to sequentially traversing a number of coarse placements is to generate and investigate different coarse placements in *parallel* This is the basic notion of Heuristic Spanning. Essentially, a heuristic algorithm is used to generate a number of grossly different but plausible placements at the same time on different processors. These are evaluated by another heuristic procedure to produce an interim "goodness" measure with which the different interim placements can be compared. One of the interim placements is chosen to be annealed further at lower temperatures to complete the full process. Fig. 1 depicts the basic process of Heuristic Spanning.

A key point is that the heuristic algorithm that runs in parallel must be much faster than Simulated Annealing, so that a reasonable speed-up can be achieved. The remainder of this section presents one Heuristic Spanning technique.

3.1. *Spanning the Search Space*

The first step of the Heuristic Spanning approach is to divide up the search space. The Min-Cut placement algorithm [3], [7], [8] used in ALTOR [181 (a standard cell placement and routing package developed at the University of Toronto) provides a convenient basis for a search space division approach. ALTOR has been measured to be about twenty times faster than SALTOR, our Simulated Annealing-based uniprocessor placement program.

The Min-Cut placement algorithm recursively subdivides a placement while minimizing the number of wire crossings at each division line. Typically, an iterative improvement partitioning algorithm such as Kemighan-Lin [141 or Fiduccia Matheyses [9] is used to do the minimization. In ALTOR, a constructive initial partitioning step was introduced to aid the Fiduccia-Matheyses [9] iterative improvement, for the *first* division step. The constructive algorithm builds one of the subdivisions of the circuit by sequentially adding the cells that are most connected to those that have already been chosen, starting with a seed cell. Experience using different seeds has shown that they have a marked effect on the quality of the final placement, yet there appears not to be a way, short of exhaustive searching, to choose the seed that will result in the best final placement. One way to solve this combinatorially difficult problem is to run the entire algorithm several times with different seeds and choose the best final placement. This is similar to the idea of using multiple random starts for an iterative improvement algorithm [2 11, but it is better because the seeds are chosen in an intelligent manner in such a way as to make the initial partitions as "different" as possible. This means that different parts of the search space will be investigated, which is the fundamental premise of Heuristic Spanning.

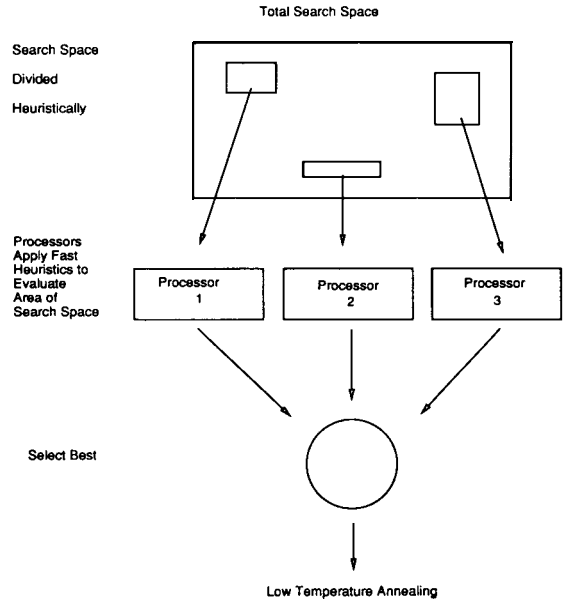


Fig. 1. The basic process of heuristic spanning.

In this context, to have *different* seeds means that the seed cells are as far apart from each other as possible. For cells to be 'far' from each other, we must define what is meant by distance. Assume that there is a set of N cells, numbered from 1 to N. Define the distance D_{ij}, between two cells i and j to be the *minimum* number of nets in the circuit that must be traversed to get from cell i to cell j.

Assume that s seed cells are required. The problem of finding different seeds is then to choose s distinct cells from the set of N such that:

$$\sum_{i=1}^{s} \sum_{j=i+1}^{s} D_{ij}$$

is maximized. Unfortunately, there are $\binom{N}{s}$ possible combinations of cells, which is a prohibitive number to investigate exhaustively. Previous to even that large computation there are $N^2/2$ of the D_{ij} to be calculated, which is excessive computation in itself. For this reason the Max-Span algorithm was developed, which has $0(sN)$ running time. It is a greedy algorithm that works well in practice [19], and is described in Fig. 2. It begins with an arbitrarily chosen first seed, and then selects the second seed as the one farthest away from the first. The next seed is chosen as the one with the greatest distance from either of the first two seeds. Subsequent seeds are selected to be as far away as possible from the seeds already chosen.

Thus, choosing a set of seed cells that are far from each other, and using initial partitions "grown" from these seeds, multiple runs of the Min-cut algorithm will investigate different areas of the *plausible* search space.

3.2. *Choosing the Best Interim Placement*

The second step of the Heuristic Spanning approach is to choose one of the interim placements to be annealed further at low temperatures. The simplest and most obvious way is to chose the interim placement with the lowest cost function. Experiments have shown (see Section

/* From a set of N cells, numbered from 1 to N, choose the s cells that are as "far" apart from each other as possible. */

Let A be the set of seed cells that have been already chosen.
Initially, set $A = \{1\}$, where cell number 1 is arbitrarily chosen.

Let B be the set of cells not yet chosen.
Initially, set $B = \{2,3,\cdots,N\}$.

The set T_j, $j = 1,\cdots,N$ is the current estimate of the minimum distance of a cell j from any of the chosen seeds. Initially, $T_j = \infty$.

The variable r is the most recently selected seed, and initially $r = 1$.

Repeat until $|A| = s$:

1. Calculate $T_j = \min(T_j, D_{rj})$ $j = 1,\cdots,N$
2. Choose cell k, as the next seed, such that $T_k = \max(T_1,\cdots,T_N)$
3. $A = A + k$ $B = B - k$ $r = k$

Fig. 2. The max span algorithm.

3.4) that there is a direct correlation between the interim and final cost functions, although the interim placement with the lowest cost function is not necessarily the one with the lowest **final cost** function. empirically however, the placement with the lowest interim cost function is always among the placements with the lower final cost function. Practically, this **means** that there must be a sufficient number of seeds processed by ALTOR to guarantee that the interim placement with the lowest cost function will achieve a **final** cost function as good as what would have been achieved by regular Simulated Annealing. No method has yet been developed to **ensure** that this occurs, but in practice ten seeds have been observed to be sufficient, as will be shown in Section 3.4.

3.3. Low Temperature Annealing

The third step of Heuristic Spanning is to anneal the interim placements at low temperatures. The crucial question here is to decide the best temperature at which to begin the annealing. If the temperature is too high then unnecessary work is done. If it is too low then the final cost function will not be as low as that for regular Simulated Annealing. The following method has been used to determine the starting temperature:

1) Determine the cost of the interim placement.
2) In a regular Simulated Annealing run, obtain a table of cost function versus temperature.
3) Determine which temperature of the full Simulated Annealing run has the closest cost function to the interim cost function. Choose that temperature as the starting temperature.

It is of course infeasible to do this matching within the approach itself, since that would mean doing a regular Simulated Annealing run every time-defeating the purpose of speeding up the process in the first place. However, the resulting temperature has been found to be constant with respect to circuit size. For all the test circuits, using the above method, the starting temperature was found to be 39 degrees, where degrees in this case have cost function units. This number is a function of the particular constants chosen in our cost function. In the regular Simulated Annealing process we used, there were 27 temperatures, and 39 was the 12th temperature, with an acceptance ratio of about 6 percent. Lower temperatures were tried, but they produced progressively higher final cost functions.

Another possibility, which we have not yet implemented, is to determine the starting temperature dynamically with every run, making use of the equilibrium characteristics of Simulated Annealing. To *measure* the temperature of a given placement, simply start at any temperature and generate several hundred moves (depending on circuit size) on the circuit, something which can be done in several seconds. The moves should not actually be accepted, but the total net change of the cost function should be recorded. If the net change is negative then the temperature should be higher, and if positive, the temperature should be lower. Using this approach a binary search can be done to quickly converge to the correct starting temperature.

3.4. *Results*

The Max-Span algorithm was implemented and used to produce ten seeds and subsequently ten interim placements (using the min-cut placement program, ALTOK) for each of six test circuits. Five of these circuits were industrial circuits provided by Bell Northern Research Ltd. The other circuit was taken from a gate array designed by the University of Toronto Microelectronic Development Centrc. Each of these placements was further annealed beginning at temperature 39. In all cases, more than one of the interim placements achieved a cost function **close** to that of the regular Simulated Annealing process (a definition of "close" is given below). Table I shows the interim and final cost function for an 118%cell circuit and the percentage difference between the final cost function and the average final cost of five regular Simulated Annealing runs. It also shows that the rank (i.e., the position in sorted order) of the interim and final cost are closely correlated, an empirical justification for using the interim cost function as a basis for choosing which placement is selected for low temperature annealing. Choosing the first seed for further annealing (as the one with the lowest interim cost) results in a placement with a cost within 0.6 percent of that achieved by the regular Simulated Annealing process.

Table II summarizes the results for all the sample circuits. It gives the lowest cost function achieved by the ten interim placements after the Min-Cut algorithm, the final cost function after annealing of the placement with the lowest interim cost function, the average and standard deviation of the final cost function of live regular Simulated Annealing runs, and the number of interim placements that were within one standard deviation of the average. This criterion was chosen because one standard deviation encompasses a significant percentage of all samples of the distribution. This last column is our method of comparing the quality of Heuristic Spanning with that of regular Simulated Annealing. Note that no fewer than two of the final placements were within one standard deviation of the average. In addition, the interim placement that would

TABLE I
LOW TEMPERATURE ANNEALING FOR 1188-CELL CIRCUIT

Interim Cost	Rank	Final Cost	Rank	% Difference from regular SA
97003	1	79978	3	+0.6%
97320	2	77979	1	-1.9%
98797	3	78869	2	-0.8%
98998	4	81342	5	+2.3%
101254	5	81734	6	+2.7%
102757	6	81161	4	+2.1%
103637	7	84511	7	+6.3%
104167	8	85027	9	+6.9%
105362	9	84539	8	+6.3%
109104	10	89613	10	+12.7%

TABLE II
HEURISTIC SPANNING RESULTS FOR ALL CIRCUITS

Circuit	# Cells	Lowest Interim Cost	Final Cost	Average of 5 Reg. S.A. Runs	Standard Deviation	# Within 1 S.D.
BNR Cct E	446	37497	29400	28186	461	2
Gate Array	590	45375	35269	35696	417	2
BNR Cct D	856	71354	57594	59105	1065	6
BNR Cct C	1188	97003	79978	79509	763	3
BNR Cct B	1515	190018	156884	157928	4176	7
BNR Cct A	1795	216253	188003	186585	1519	2

have been chosen in the Heuristic Spanning process is close to or better than the average of five regular Simulated Annealing runs. From these results, it is clear that Heuristic Spanning works well, and achieves quality equivalent to that of regular Simulated Annealing.

3.5. *Performance*

The performance improvement of Heuristic Spanning over regular Simulated Annealing come from two sources:

1) The algorithm itself is faster than the high-temperature portion of Simulated Annealing when run on a uniprocessor.
2) Heuristic Spanning can be easily sped up using multiple processors.

The number of interim placements that is produced in Heuristic Spanning must be sufficient to guarantee that the interim placement with the lowest cost function will have a final cost function as good as regular Simulated Annealing. Empirically, IO interim placements have been shown to be sufficient for the 6 test circuits.

Table III gives the performance improvement of Heuristic Spanning over regular Simulated Annealing. For each circuit it gives the time for ALTOR to produce one placement, the time to execute a full Simulated Annealing run and the portion of that time which corresponds to the high temperature phase. The last column gives the speed-up of Heuristic Spanning run on one processor (i.e., one processor generates all ten interim placements) compared with the high temperature portion of regular Simulated Annealing. It is between 1.5 and 2.5 times faster for all the test circuits.

If Heuristic Spanning were run using 10 processors (each generating an interim placement), the speed-up over the one processor Heuristic Spanning time would be almost exactly 10. This is because the Max-Span algorithm takes negligible time to compute and there is no other serial component in the calculation. Thus the speedup of l0 processor Heuristic Spanning over uniprocessor Simulated Annealing at high temperature is simply a factor of 10 better than the figure given in the last column of Table III. This results in factors of improvement from 15 to 25 times, using only 10 processors.

TABLE III
SPEED-UP OF HEURISTIC SPANNING OVER SIMULATED ANNEALING

Circuit Name	# Cells	ALTOR Execution Time (s)	SALTOR Execution Time (s)	High Temp Portion (s)	Speed-Up
BNR Cct E	446	131	7270	3231	2.5
Gate Array	590	232	10578	4701	2.0
BNR Cct D	856	279	12180	5413	1.9
BNR Cct C	1188	402	21224	9432	2.4
BNR Cct B	1515	815	26890	11951	1.5
BNR Cct A	1795	1024	34347	15265	1.5

This marked increase in speed of the multiprocessor algorithm over uniprocessor Simulated Annealing is due to the combination of a faster algorithm and the use of multiple processors. It is possible that when the parameters of Simulated Annealing such as the cooling rate and the number of acceptances per temperature are better known, these figures will not be quite so dramatic. At present, since the properties of Simulated Annealing are not well understood (though some recent work [12] has shed some light on the subject), it is reasonable that a more intelligent heuristic will be more efficient.

IV. Low Temperature: Section Annealing

The low temperature phase of Simulated Annealing has several appealing factors that argue in favor of using it directly in a parallel implementation:

1) The low acceptance ratio affords a direct parallelism since data is changed infrequently.
2) The small move size at low temperature provides locality that is useful in creating independent parallel tasks.
3) Intuitively, the random "hunting" for good moves that goes on at low temperature makes sense, especially in the context of a multiprocessor environment where it is easy and cheap to add more processors.

Section Annealing begins with an interim placement for which the high temperature placement or equivalent has already been performed. This placement is divided up geographically, with the geographic areas and the cells contained in those areas assigned to separate processors. Each processor then generates Simulated Annealing-style moves, in parallel, for the cells which it is assigned, and tests those moves for acceptance. If a move is not accepted then no further work is done. If a move is accepted, then the accepting processor transmits the move tc the other processors so they can maintain a consistent view of cell positions. Fig. 3 depicts the basic process of Section Annealing.

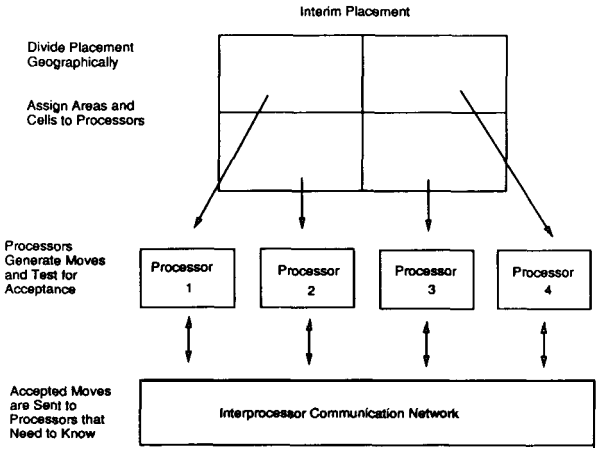


Fig. 3. The basic process of Section Annealing

4.1. *Multiprocessor Algorithm Design*

There are a number of design trade-offs involved in Section Annealing.

Asynchronous versus Synchronous Moves:

There are two choices concerning the synchronization of the processors as they generate moves: each processor can either stop after every move and wait for the others to finish their moves (the synchronous case) or continuously generate moves without regard to the state of the other processors (the asynchronous case). The synchronous case allows a central database of cell positions to be maintained in a consistent fashion and thus permits Simulated Annealing to be reproduced exactly [16]. Unfortunately, since different moves require different amounts of time to be evaluated (i.e., determine the change in cost function), synchronization means that processors will sit idle waiting for the slowest to finish, which can be a severe performance penalty. Also, the task of keeping the database consistent when several moves are accepted at once (if this consistency is desired) takes a great deal of time.

The asynchronous case allows the processors to run faster, unhindered by synchronization, but introduces an error in the process because the cell position database(s) as *seen* by the processors will not be consistent. This occurs when one processor changes the position of a cell while another is doing a cost function calculation using it.

Since we require the greatest speed possible, we chose the asynchronous case. The effects of the error induced are discussed further, in Section 4.2.

Central versus Separate Databases:

The database containing the cell locations can either be in one central location (which all processors read and update directly) or each processor could have its own copy of the data. In the case of a central database, it is easier to make changes to the cell positions because the changes are only made once. During such changes, however, the database may be in a dramatically inconsistent state (broken linked lists and other data structures) so some kind of locking of data structures would be necessary. A central database can also be a bottleneck because it is accessed by every processor for every move. Local caches may alleviate this, however, at the price of requiring some form of cache consistency.

Having separate databases in each processor means that global communication is required only when a processor makes a move (changing the database) rather than at every move. This is good because Section Annealing is done at temperatures that have low acceptance ratios, and thus global accesses are infrequent. However, since there are multiple copies of the same information, there will always be times when different copies are inconsistent-causing erroneous moves to be made.

We chose the separate database implementation, so as to avoid the central bottleneck, keeping in mind that the misinformation would have to be dealt with at some point. This choice was also influenced by the architecture of our prototype multiprocessor, which has large local memories.

Move *Generation:*

The types of moves used in Section Annealing are basically a subset of those in [23]: single-cell displacements and two-cell exchanges, with range-limiting. If a displacement causes a cell to move beyond its processor's geographically assigned area, then responsibility for generating moves for that cell is transmitted to the processor assigned to the new area. This is called a *displacement responsibility transfer.*

The only difference from [23] is that exchange-type moves are only generated among the cells assigned to a particular processor. Exchanges of cells between *two* processors would necessitate interlocking to ensure that an exchanged cell had not already been moved by the remote processor, and would be difficult to do. The intra-processor exchange restriction of the search space is not significant since cells are allowed to displace across processor boundaries and displacements outnumber exchanges by a ratio of 5 : 1 [23]. Empirically, this restriction has resulted in no loss of convergence.

Geographic Division:

The task of assigning geographic areas and the cells in those areas to processors is non-trivial. The division technique must allow the assignment of arbitrary numbers of cells to processors for workload balancing (see Section 4.4) and ensure that the areas are as square as possible to reduce the number of displacement-responsibility transfers: if the boundaries are allowed to be an arbitrary rectilinear shape, more perimeter will be exposed to other processors' areas, increasing the likelihood that they will have to be informed of the local processor's moves. Division is accomplished by "sweeping" out manually designated areas (i.e., areas defined by the CAD programmer in a table, rather than by some automatic technique) until the right number of cells is collected [19].

4.2. *Misinformation*

The principle drawback of using separate cell databases is that they can be in inconsistent states. This occurs be-

tween the time that one processor moves a cell and when it informs the other processors of that move. If another processor makes a move based on the cell's location during that time, we call this a *misinformed move*, and say that placement then contains some *error*. Total error is defined as the difference between the sum of the changes in cost function as viewed by the processors and the actual change in cost function when the data are made consistent. We need to know what the effect of the misinformation is, and how much error the process can withstand. Grover [10] claims that the most error that can be withstood is about half of the current temperature.

In our first simple implementation of Section Annealing, we were able to do some experiments that gave valuable insight into the effect of the error. In this implementation, the separate databases were updated only after some fixed number (X) of moves were generated by each processor. By changing X, we were able to observe the convergence properties of Section Annealing with varying amounts of error. With a 552-cell circuit, we found that if the average number of moves accepted without being broadcast to all processors was less than roughly 10, then the Section Annealing process converged to the same final cost function as regular Simulated Annealing. If more than 10 moves were accepted (by all of the processors) then the cost function became unstable and was observed to increase monotonically, rather than decrease. These basic observations on the effect of misinformation on convergence bode well for the implementations described below, because we anticipate far fewer than 10 moves will be accepted without every processor being informed.

We note that this was a limited experiment that provided a rough idea of Section Annealing's tolerance for error. Further experimentation and analysis is required to determine these properties for varying sizes of circuits and under different temperature conditions.

One further effect of misinformation is discussed below in Section 4.4.

4.3. *Implementation of Section Annealing*

The first full implementation takes the obvious approach to maintaining consistent databases in each of the processors: as soon as any processor accepts a move, it immediately broadcasts it to all of the processors. This is called the *Full-Broadcast* mode. Since Section Annealing will deal with no more than $P = 20$ processors, and will be used at acceptance ratios of no more than $A = 0.06$, then on average no more than $P \times A = 1.2$ moves will be accepted during one move period. Since this number is much less than 10, we expect to get convergence equivalent to that of regular Simulated Annealing, and indeed this has been observed experimentally, as shown in Table IV. This table gives the final cost function of Full-Broadcast Section Annealing for one to five processors, for each of the five test circuits. There is no significant difference between the uniprocessor cost function and the 2-5 processor cost functions.

TABLE IV
CONVERGENCE OF SECTION ANNEALING USING 1–5 PROCESSORS

Circuit	1 Processor	2 Processors	3 Processors	4 Processors	5 Processors
Gate Array	35036	36282	35582	35534	35497
BNR D	58468	58592	58450	58043	58494
BNR C	78305	80332	77885	78403	77393
BNR B	151329	155609	154261	152575	152332
BNR A	181485	183577	184327	183072	180554

In the Full-Broadcast approach, the principle performance degradation is due to each of the processors spending time updating their local databases *(not* the time to transmit the move) when informed of extra-processor moves. In fact, every processor does not really need to know about every move made elsewhere. A processor only needs to know about the motion of a cell when:

1) It contains a cell connected to the moved cell (to calculate wire length correctly), or
2) It contains at least one cell less than the range window distance away from either the old or new position of the cell (to calculate overlap penalties correctly), or
3) It contains a row whose total width has changed due to the move (so it can calculate row width penalties correctly).

Rather than broadcast every move, in the need-to-know approach only the processors that are in one or more of these categories for a given move are informed. There is computation required, however, to determine which processors are in these categories. For this method to be useful, the time spent in that calculation must be less than the total amount of computation saved.

Measured results indicate that, at the higher temperatures in the early phase, this is not true-basically all of the processors "need to know" about every move. At the lower temperatures, however, the number of moves sent to other processors is reduced by as much as 50 percent, a significant saving. Define f_N as

$$f_N = \frac{\#\text{ Moves sent to other processors in Need-To-Know scheme}}{\#\text{ Moves sent to other processors in Full-Broadcast scheme}}$$

Fig. 4 is a plot of f_N versus temperature stage, using 5 processors. (A temperature stage is the work done at each unique temperature. There are 27 temperature stages in our version of regular Simulated Annealing; Section Annealing begins at stage number 13.) It is clear that at some point, depending on the overhead involved, it is worthwhile switching to the need-to-know approach.

4.4. *Observations and Difficulties*

There are several interesting observations and difficulties encountered in implementing Section Annealing.

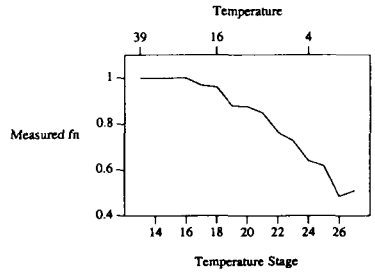


Fig. 4. f_N versus temperature stage, for 590-cell circuit using 5 processors

Acceptance Ratio:

In running Section Annealing, the acceptance ratio was observed to be an increasing function of the number of processors. This is due to the fact that the amount of misinformation, or error, increases as the number of processors increases. If a move is made based on misinformation then it is likely the move is bad (i.e., increases the cost function) since most moves are bad. This move would not have taken place in regular Simulated Annealing. Furthermore, since we have observed convergence of Section Annealing to the same final cost function as regular Simulated Annealing, then further good moves have to be made to make up for the bad move. Thus the acceptance ratio increases in the presence of misinformation, which is an increasing function of the number of processors.

Cell Balance:

On occasion, all of the cells assigned to one processor were observed to *vacate* that processor and the geographic area it was assigned. This is clearly an effect of the multiprocessor algorithm, since it never happens in uniprocessor Simulated Annealing. The effect can be explained as follows. The number of moves made by a processor is directly proportional to the number of cells it is originally assigned. (This keeps the multiprocessor algorithm comparable to the uniprocessor version.) Due to the random nature of Simulated Annealing it is possible that the number of cells in a processor will dip below some *critical* amount resulting from displacement-responsibility transfers. Here, more moves are generated per cell for the cells remaining in that processor, making it more likely a move will be accepted for each cell. Since more of each cell's circuit neighbors have already left the processor, it becomes more likely that remaining cells will also move out. Thus a kind of positive feedback occurs and soon all cells leave the processor.

Fortunately, the effect is easy to cure: simply rebalance the cells among processors whenever the number in any procesor dips below 75 percent of the original assignment. This has been observed to solve the problem and happens infrequently enough to have a negligible effect on performance.

Workload Balance:

Since cells in circuits are different from each other, it takes varying amounts of time to calculate a change in cost function for different cells. If a set of "slow" cells are grouped together in one processor, then it will take a longer time for that processor to complete the same number of moves as another processor. This means that processors will be idle waiting for the slowest processor to finish, decreasing performance.

To correct this problem, the speed at which each processor is executing moves is measured at each synchronization point (after about every 2000-3000 moves is made in each processor). The number of cells and area of geographic responsibility in each "slow" processor are reduced and the number in each "fast" processor increased in a time-balancing algorithm [19]. This approach corrects the time imbalance that occurs when one processor is more than 10 percent slower than the fastest processor.

To completely correct the remaining 10 percent imbalance, all processors terminate when the first processor finishes, and the moves that are missed are evenly distributed over all processors at the end of the temperature stage.

4.5. *Performance*

Section Annealing has been successfully implemented on a six-processor prototype. One of the processors is dedicated to operating system functions, so we have five usable processors.

As mentioned above, Section Annealing converges to the same final cost function as regular Simulated Annealing. The timing and speed-up results for the Full-Broadcast case is given in Table V, for an 856-cell circuit. It achieves a speed-up of 4.3 using 5 processors, where speed-up for n processors is given by $S_n = T_1 / T_n$ and T_n is the execution time using n processors. Table VI summarizes the results for all of the test circuits, ranging in size from 552 cells to 1795 cells. Similar speed-up results were obtained for all circuits. Circuit BNR Chad a worse speed-up because it exhibited a higher acceptance ratio at the lower temperatures, The reason for this is not known.

The timing and speed-up results for need-to-know section annealing are given for one to five processors on an 856-cell circuit in Table VII. The speedup is given relative to the uniprocessor Full-Broadcast case. As discussed above, the overhead incurred in determining which processors need to know more about a move is greater than the work saved at the higher temperatures. Thus, the speed-up is only 4.1 using 5 processors; less than that of the Full-Broadcast case for the same circuit. However, since need-to-know is worthwhile at lower temperatures, an adaptive approach which switches from Full-Broadcast to need-to-know will provide a performance improvement

Fig. 5 is a plot of the speed-up versus the number of processors for both the Full-Broadcast and Need-To-Know approaches, for the 856-cell circuit.

A model developed in [19] takes into accountall of the properties of Full-Broadcast Section Annealing discussed here, and allows the prediction of performance for up to 10 processors. Table VIII gives the predicted lo-processor results for all the test circuits. We expect a speed-up

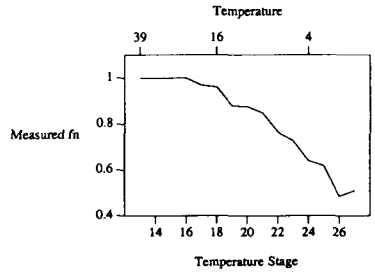


Fig. 4. f_N versus temperature stage, for 590-cell circuit using 5 processors

Acceptance Ratio:

In running Section Annealing, the acceptance ratio was observed to be an increasing function of the number of processors. This is due to the fact that the amount of misinformation, or error, increases as the number of processors increases. If a move is made based on misinformation then it is likely the move is bad (i.e., increases the cost function) since most moves are bad. This move would not have taken place in regular Simulated Annealing. Furthermore, since we have observed convergence of Section Annealing to the same final cost function as regular Simulated Annealing, then further *good* moves have to be made to make up for the bad move. Thus the acceptance ratio increases in the presence of misinformation, which is an increasing function of the number of processors.

Cell Balance:

On occasion, all of the cells assigned to one processor were observed to *vacate* that processor and the geographic area it was assigned. This is clearly an effect of the multiprocessor algorithm, since it never happens in uniprocessor Simulated Annealing. The effect can be explained as follows. The number of moves made by a processor is directly proportional to the number of cells it is originally assigned. (This keeps the multiprocessor algorithm comparable to the uniprocessor version.) Due to the random nature of Simulated Annealing it is possible that the number of cells in a processor will dip below some *critical* amount resulting from displacement-responsibility transfers. Here, more moves are generated per cell for the cells remaining in that processor, making it more likely a move will be accepted for each cell. Since more of each cell's circuit neighbors have already left the processor, it becomes more likely that remaining cells will also move out. Thus a kind of positive feedback occurs and soon all cells leave the processor.

Fortunately, the effect is easy to cure: simply rebalance the cells among processors whenever the number in any procesor dips below 75 percent of the original assignment. This has been observed to solve the problem and happens infrequently enough to have a negligible effect on performance.

Workload Balance:

Since cells in circuits are different from each other, it takes varying amounts of time to calculate a change in cost function for different cells. If a set of "slow" cells are grouped together in one processor, then it will take a longer time for that processor to complete the same number of moves as another processor. This means that processors will be idle waiting for the slowest processor to finish, decreasing performance.

To correct this problem, the speed at which each processor is executing moves is measured at each synchronization point (after about every 2000-3000 moves is made in each processor). The number of cells and area of geographic responsibility in each "slow" processor are reduced and the number in each "fast" processor increased in a time-balancing algorithm [19]. This approach corrects the time imbalance that occurs when one processor is more than 10 percent slower than the fastest processor.

To completely correct the remaining 10 percent imbalance, all processors terminate when the first processor finishes, and the moves that are missed are evenly distributed over all processors at the end of the temperature stage.

4.5. *Performance*

Section Annealing has been successfully implemented on a six-processor prototype. One of the processors is dedicated to operating system functions, so we have five usable processors.

As mentioned above, Section Annealing converges to the same final cost function as regular Simulated Annealing. The timing and speed-up results for the Full-Broadcast case is given in Table V, for an 856-cell circuit. It achieves a speed-up of 4.3 using 5 processors, where speed-up for n processors is given by $S_n = T_1 / T_n$ and T_n is the execution time using n processors. Table VI summarizes the results for all of the test circuits, ranging in size from 552 cells to 1795 cells. Similar speed-up results were obtained for all circuits. Circuit BNR Chad a worse speed-up because it exhibited a higher acceptance ratio at the lower temperatures, The reason for this is not known.

The timing and speed-up results for need-to-know section annealing are given for one to five processors on an 856-cell circuit in Table VII. The speedup is given relative to the uniprocessor Full-Broadcast case. As discussed above, the overhead incurred in determining which processors need to know more about a move is greater than the work saved at the higher temperatures. Thus, the speed-up is only 4.1 using 5 processors; less than that of the Full-Broadcast case for the same circuit. However, since need-to-know is worthwhile at lower temperatures, an adaptive approach which switches from Full-Broadcast to need-to-know will provide a performance improvement

Fig. 5 is a plot of the speed-up versus the number of processors for both the Full-Broadcast and Need-To-Know approaches, for the 856-cell circuit.

A model developed in [191 takes into account all of the properties of Full-Broadcast Section Annealing discussed here, and allows the prediction of performance for up to 10 processors. Table VIII gives the predicted lo-processor results for all the test circuits. We expect a speed-up

cuits for use in this work. Tom Blank provided a helpful review of this paper.

REFERENCES

[1] P. Banerjee and M. Jones, "A parallel simulated annealing algorithm for standard cell placement on a hypercube computer," in *Proc. IC-CAD* 86, pp. 34-37, Nov. 1986.

[2] D. R. Blythe "Master/slave TUNIS: A multiprocessor operating system," M.Sc. thesis, Dep. Computer Science, University of Toronto, 1986.

[31 M. A. Breuer, "Min-cut placement," *J. Design automation Fault-Tolerant Computing,* pp. *343-362,* Oct. *1977.*

[41 A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macro-cells," in *Proc. ICCAD 86,* pp. *30-33, Nov. 1986.*

l51 D-J. Chyan and M. A. Breuer, "A placement algorithm for array processors," in *Prac. 20/h Design Automatian* **Conf ,** pp. 182-188. June 1983.

161 l. R. Cordy and R. C. Holt "Specification of concurrent Euclid," Computer Systems Res. Group Tech. Rep. CSRG-133, University of Toronto, Aug. 1981.

[71 L. I. Corrigan, "A placement capability based on partitioning," in *Proc. I6th Design Automution* **Conf ,** pp. 406-4 13, June 1979.

181 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.

I91 C. M. Figuccia and R, M. Matheyses, "A linear time heuristic for improving network partitions," in *Prac. 19th Design Automation Conf.* pp. 175-181, June 1982.

[10] L. K. Grover, "A new simulated annealing algorithm for standard cell placement," in *Proc. ICCAD 86,* pp. 378-380, Nov. 1986.

[11] C. A. R. Hoare "Monitors: An operating system structuring concept," *Commun. ACM, vol. 17,* no. 16, pp. *547-557,* Oct. 1974.

[12] M. D. Huang, F. Romeo and A, Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proc. IC-CAD 86,* pp. *381-384, Nov. 1986.*

[13] ,&. Iosupovicz, C, King, and M. A, Breuer, "A module interchange placement machine," in *20th Design automation Conf,.* pp. 17 1- 174, June 1982.

[14] B. W. Kernighan and S. Lin "An efIicient heuristic procedure for partitioning network graphs," *Bell Syst. Tech. J.,* pp. *29 1-307,* Feb. 1970.

[15] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science, vol.* 220, no. 4598, pp. 671-680, May 1983.

[16] S. A. Kravitz, "Multiprocessor-based placement by simulated annealing," SRC-CMU Centre for Computer Computer-Aided Design, Research Rep. CMUCAD-86-6, 1986.

[17] S, A, Kravitz and R. A. Rutenbar, "Multiprocessor-based placement by simulated annealing," in *Proc, 23rd Design Automation Canf.,* pp. 567-573, June 1986.

[18] J. S, Rose, W, M. Snelgrove, and Z. G. Vranesic, "ALTOR: An automatic standard cell layout program," in *Proc. Canadian Conf. VLSI,* pp. 168-173, Nov. 1985.

[19] J. S. Rose, "Fast, high quality VLSI placement on a MIMD multiprocessor," Ph.D. dissertation, Dep. of Electrical Engineering, University of Toronto, 1986; also Computer Systems Res. Inst. Tech. Rep. # 189.

[20] J. S. Rose, D. R, Blythe, W. M. Snelgrove, and *Z. G.* Vrancsic, "Fast, high quality VLSI placement on an MIMD Multiprocessor," in *Proc. /CCAD 86,* pp. 42-45, Nov. 1986.

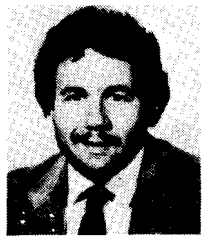
[21] C, Rowen and J. J. Hennessy, "SWAMI: A flexible logic implementation svstem," in *Proc.* 22nd *Design Automatian* **Canf ,** pp. *169-*175, June, 1985.

[22] R. A. Rutenbar and S. A. Kravitz, "Layout by annealing in a parallel environment," in *Proc. Int. Conf. Compu/er Design: VLSI in Cam*purer,y *(ICCD),* pp. 434-437 Oct. 1986.

[23] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf placement and routing package," *IEEE J. Solid-State Circuits, vol. SC-20,* pp. 510-522. Apr. 1985.

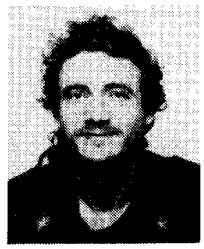
[24] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A new standard cell placement and global routing package," in *Proc. 23rd Design automation Conf,* pp. 432-439, June 1986.

[25] K. Ucda, T. Komatsubara, and T. Hosaka, "A parallel processing approach for logic module placement," *IEEE Trans. Computer-Aided Design, vol.* CAD-2, pp. 39-47, Jan. 1983.



Jonathan Rose (S'79-M'86) received the B.A.Sc. degree in Engineering Science in 1980, and the M.A.Sc. and Ph.D. degrees in electrical engineering in 1982 and 1986, respectively, from the University of Toronto.

During the summer of 1983, Rose was with the Bell-Northern Research Ltd., Ottawa, in the Integrated Circuits CAD/CAM group. He is a visiting Post-Doctoral Scholar in the Computer System Laboratory, Stanford University, CA. His research interests include CAD for placement and routing, parallel processor architectures and applications, and combinations of the two.



W. **Martin Snelgrove (S'75-M'81)** *was* born in Kitwe, Zambia, in October 1954. He received the B.A.Sc., degree in chemical engineering in 1975, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1977 and 1982, respectively.

In 1982 he worked at the Institute National de Astrofisica, Optica y Electronics, Tonantzintla, Mexico, as a visiting investigator. Since then he has been at the University of Toronto as an Assistant Professor. He is involved in research projects in the University's Computer Systems Research Institute and its VLSI Research Group, primarily in the areas of CAD on multiprocessors and high-frequency integrated filters. A I986 paper co-authored with A. Sedra was the winner of the 1986 IEEE Circuits and Systems Society Guillemin-Cauer Award.

Zvonko G. Vranesic (S'67-M'68-SM'84) received the B.A.Sc., M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Canada, in 1963, 1966 and 1968, respectively.

From 1963 to 1965 he was with the Northern Electric Company Ltd., Bramalea, Ont., Canada. In 1968 he joined the Faculty of the Departments of Electrical Engineering and Computer Science at the University of Toronto, where he is now a Professor. During 1977 to 1978 and 1984 to 1985 he was a Senior visitor in the Computer Laboratory at the Universily of Cambridge, England, and in the Institut de Programmation at the University of Paris 6, France. His research interests include computer architecture, fault tolerant computing, local area networks and many-valued switching systems. He was the Chairman of the 1973 International Symposium on Multiple-Valued Logic and the Technical Program Chairman of the 6th International Symposium on Multiple-Valued Logic.