Parallel algorithms for chip placement by simulated annealing

by F. Darema
S. Kirkpatrick
V. A. Norton

We explore modifications to the standard simulated annealing method for circuit placement which make it more suitable for use on a shared-memory parallel computer. By employing chaotic approaches we allow the parallel algorithms to deviate from the algorithm defined for a serial computer and thus obtain good execution efficiencies for large numbers of processors. The qualitative behavior of the parallel algorithms is comparable to that of the serial algorithm.

1. Introduction

In engineering design, a wide range of problems require optimizing some figure of merit which depends upon a great many quantitative design parameters. Examples are sequencing problems such as the traveling salesman problem, or circuit placement and wiring in computer-aided electronic design. Solution of these problems is difficult because there are far too many degrees of freedom to permit exhaustive search for an optimum, and in many cases (the so-called NP-complete problems) no algorithms are known which will determine the exact optimum with significantly less work than exhaustive search. Instead, heuristics are employed, such as iteratively searching through local rearrangements of a design in which one or a few parameters are changed in each rearrangement. Recently, the method of *simulated annealing* (SA) [1] has proven successful as a common nondeterministic approach to most of these problems. This method obtains good solutions, even when conventional methods quickly get trapped in local sub-optima. A state-of-the-art overview of the theory and applications of SA is given in [2].

In the SA method, an analogy is made between the states of a physical system (e.g., a *solid*) and the configurations of the design being optimized. The design parameters being varied are equated with atomic positions in the solid, and the energy of the solid is identified as the objective function being optimized for the design. Through the analogy, a temperature is defined as a control parameter. In optimization by simulated annealing, the temperature controls the probability that rearrangements which make the design worse will be accepted in order to have a more exhaustive search. Just as in growing perfect crystals from molten mixtures by annealing them at successively lower temperatures, the optimization process using SA proceeds by searching at successively lower temperatures, starting at a high temperature at which nearly random rearrangements are accepted (the *melted* state). Although this procedure is computationally laborious, it is no more so than the best heuristics previously available for many design problems, and in most cases finds the best solutions which are known.

Currently a variety of MIMD (multiple-instructions-multiple-data) computer architectures are being proposed as a means of providing increased computational power. The characterization of these architectures is that they employ multiple processors interconnected and cooperatively working to execute the same program. Such *parallel*

architectures require not only new ways of performing the computations and new ways to execute algorithms, but in many cases new ways of thinking about the algorithms. In particular we have in mind parallel architectures that utilize *shared memory* [3] as a means of synchronization and communication between the parallel processors. The shared memory contains, for example, the description of the problem under study and the current version of the design being optimized. The effectiveness of the parallel algorithm, and the parallel hardware, is often measured by the speedup, defined by dividing the elapsed time required to solve a problem on P processors into the time required to execute the algorithm on a single processor.

In this paper we explore deviations from the form of SA appropriate for a single processor (we call this the *serial algorithm*) which enhance the performance of SA on parallel hardware. Convergence of simulated annealing can be proved in the serial algorithm, but the deviations we consider will invalidate the usual proofs. Thus we shall consider the quality of the solutions found using parallel SA, as well as the efficiency of doing the calculation in parallel.

To model machine architectures with various numbers of processors dedicated to this problem, we employ a software environment for parallel execution (VM/EPEX [4]) that runs on an IBM 3081 multiprocessor. In this environment, each processor is emulated by a *virtual machine*, a separate process created under the VM/SP operating system. Shared memory for communication between emulated processors is provided by standard VM system facilities.

2. The serial SA algorithm for chip placement

First we describe the application of simulated annealing to a simple version of the chip (or circuit) placement problem. Our chips (also referred to as *books* or *cells*) are all identical in size, and situated in a regular array of allowed positions. The techniques we describe are also appropriate for placing standard-sized circuits on regularly structured gate array chips, and can be extended to describe more complicated situations. All the connections required by the design are specified in a *netlist*, or list of nets, where the ith net, connecting n chips, would be defined by the statement that the output of chip $j(i, 1)$ is connected to specified inputs on chips $j(i, 2), \cdots, j(i, n)$. We also specify the coordinates of each chip, $x(j)$, $y(j)$. For this example, we suppress details of chip orientation, location of specific input or output pins on the chip boundary, etc., treating the chips as pointlike.

To rearrange the placement, we pick a pair of chips at random and consider the effect of interchanging them. The desirability of the exchange is evaluated by calculating the change it would cause in a cost function. The cost function is also called the *score* for the problem or, using the annealing metaphor, the *energy*; we use the latter term in this paper. The two principal factors in the energy E are the length of wire L required to complete all the nets in the netlist, and the congestion C which may occur if a placement calls for a greater density of wires in some region than the intended package can provide. We define below one possible measure of congestion to use. The two factors must be weighted and combined into a single number to be used as an objective criterion for accepting or rejecting trial exchanges. Thus, introducing a weight factor W, we have

$$E = L + WC$$

for the calculation of the energy.

The length of a single connection $l(j, k)$ between two chips j and k is calculated in the *Manhattan metric*, defined by

$$l(j, k) = |x(j) - x(k)| + |y(j) - y(k)|.$$

We approximate the length of a multichip net by its lower bound, the half-perimeter of a rectangle tightly bounding all the chips on the net:

$$l(i) = \max_k \{x[j(i, k)]\} - \min_k \{x[j(i, k)]\} + \max_k \{y[j(i, k)]\} - \min_k \{y[j(i, k)]\},$$

and sum over nets i to obtain L:

$$L = \sum_{\text{nets } i} l(i).$$

Congestion can be measured by counting the number of nets which must cross lines of constant x or y coordinate. Histograms $h(x)$ and $h(y)$ are convenient ways to collect this information:

$$h(x) = \sum_{\text{nets } i} \theta(x = \min_k \{x[j(i, k)]\})\theta(\max_k \{x[j(i, k)]\} - x)$$

(where the function θ is 1 if its argument is positive, 0 otherwise) and similarly for $h(y)$. Congestion problems can be identified by comparing the entries in $h(x)$ and $h(y)$ with known package capacities. For the global measure C we have used a simpler quantity,

$$C = \sum_x [h(x) - h_0]^2\theta[h(x) - h_0] + \sum_y [h(y) - h_0]^2\theta[h(y) - h_0],$$

where we have subtracted h_0, a congestion threshold. The excess over the threshold is squared to increase the penalty of the deviation.

The histograms also provide a convenient way to calculate L or changes in L, since

$$L = \sum_x h(x) + \sum_y h(y).$$

Use of the histograms makes it relatively inexpensive to calculate the change in the cost function that results from a given trial exchange of two chips.

A pair of chips is selected for exchange, and the *new energy* (E_{new}) of the proposed move is calculated. Then the difference ΔE from the *old energy* (E_{old}) is calculated: $\Delta E = E_{\text{new}} - E_{\text{old}}$. A move is always accepted when the effect of the

exchange decreases the energy; if the energy increases, the move is accepted, with a certain probability that depends on the *temperature* of the system, according to the Metropolis criterion:

If $\Delta E \leq 0$, accept the move.
If $\Delta E > 0$, accept the move with probability $P(\Delta E) = e^{(-\Delta E/T)}$,

where T is the temperature. When the move is accepted, the old energy is replaced by the new energy.

At a given temperature several exchanges are attempted; then the temperature is decreased and the procedure is repeated. Thus the system is gradually *cooled* until it has reached some cutoff (steady-state) criteria such as the following: The number of moves accepted is small and no substantial reduction of the energy is achieved.

The maximum number of attempts (*trials*) made at each temperature is such as to obtain a good statistical ensemble of trials. The number of distinct pairwise permutations on a board of N chips is $N \times (N-1)/2$; thus, the maximum number of trials we allow at each temperature is about that. In addition, another criterion is built into the algorithm: If at a given temperature a certain number of successful attempts have been made, it is decided that the state space has been searched adequately and the search stops for this temperature. In our particular problem we choose a number of successes equal to about 10% of the maximum number of exchanges as providing a good sampling of the state space.

At high temperatures uphill moves are accepted with a higher probability than at low temperatures, so at high temperatures the upper limit of moves is not reached. At the lower temperatures, however, the maximum number of trials is always reached; thus most of the computation is spent in the lower temperature region.

For a good annealing schedule, the temperature of the system has to be high enough initially so that the system becomes completely disordered (liquid phase) and no partially ordered domains remain. The rate at which the system is cooled down is an important consideration in obtaining convergence. A slower cooling rate increases the chances of obtaining good solutions at increased computational cost. Several quasi-empirical criteria have been established regarding the cooling schedule [1, 5, 6] and the parameters that are important to examine. Common tools for judging the quality of convergence of the algorithm are plots of the energy vs. temperature, or plots of the average value of the energy vs. temperature; they are referred to as *annealing curves*.

3. Model problem

We have used for our studies a two-dimensional configuration of 81 chips, arranged on a square grid with each net connecting two chips (*two-pin connections*). **Figure 1** shows the configuration of the chips and their connections when the system is in its ground state. The chips are labeled A1–A9, B1–B9, etc.

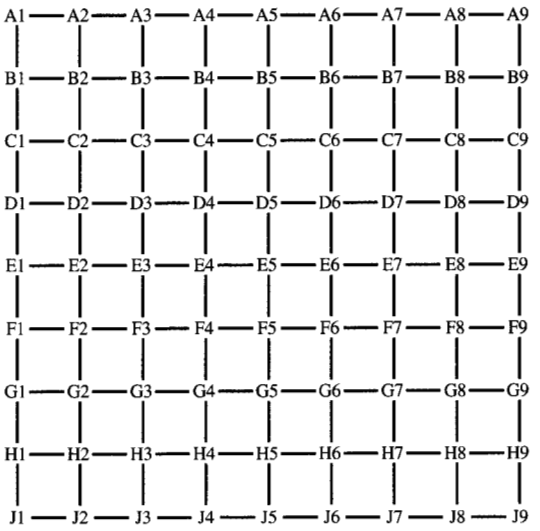

Figure 1
Ground-state configuration of chips on a rectangular grid.

This is a simplified model of actual connections in chip layouts, where the most usual case is three or more chips per net. On the other hand, our model resembles circuit connections in master-slices. However, the methods and the program we have used are general enough to handle any types of connections. The advantage of selecting such a problem is that the states of the system (ground state and excited states) are known; therefore it provides a good test bed for evaluating the algorithms themselves. Because we are interested in the performance of nondeterministic algorithms, using a problem whose solution is known makes it possible to compare the quality of solutions obtained with different algorithms.

On the basis of the algorithms discussed in the previous section, the energy for the ground state is 160 (using $h_0 = 8$), while for states differing by a transposed pair from the ground state, the energy is around 200. Scores above that and below 400 correspond to configurations which differ from the ground state by two pairs of chips transposed or configurations where a set of chips is rotated.

In all our experiments we follow a cooling schedule as suggested in [1]. The temperature is first increased until a sufficient percentage of moves is accepted (75% in our

problem); that corresponds to a temperature high enough to disorder (melt) the system. Then the cooling phase starts and the temperature is gradually lowered, by a ratio of 0.98 ($T_{new} = 0.98T_{old}$).

Following our arguments in Section 2 for obtaining a good statistical ensemble of trials, at each temperature either we attempt a maximum of $50N = 4050$ trials [that is, about $N \times (N-1)/2$ trials], or trials are made until the number of moves accepted is 10% of the maximum allowed attempts (that is, $5N$ successes). The cooling stops a) if the temperature drops below 0.01; b) if for a certain number (120) of temperatures less than $5N$ moves were accepted; or c) if a temperature has been reached at which no successful moves are made.

4. Parallel SA methods

The problem in parallel execution of the SA method is how to enable multiple processors to act cooperatively. Here we discuss two methods, and their variations, that we have implemented to execute the algorithm in parallel.

The two methods have the following in common: As in the serial algorithm, we maintain one configuration of the system; but now the various processors attempt simultaneous exchanges of disjoint pairs of chips at each temperature. The pairs are assigned dynamically; that is, we do not assign a fixed group of chips to each processor. Thus, each processor randomly picks two chips, sets a flag so that no other processor can move either of these two chips (we call this *locking* the chips), and attempts to exchange their positions. If a chip is found locked, another chip is searched for. We do not wait for the locked chip to become free, because the computation time required to select a chip and attempt to lock it is much smaller than the time needed to decide on the move, exchange the chips, and then free the locked chips. During the process of deciding on the acceptance or rejection of the move, each processor calculates locally the change in the histograms and evaluates, also locally, what the new energy would be due to the attempted move; then the (local) new energy is compared with the (shared) energy. If according to the acceptance criteria the move is accepted, the processor updates the (shared) histogram information (using the *fetch-and-add* function [4]) and replaces the (shared) energy value with its new energy. Each processor calculates the energy, and the move is accepted or rejected on the basis of information which may change during the evaluation. Thus, we introduce some chaos into the procedure of making the decision for acceptance or rejection of a move; in other words, the computation contains noise from the effects of other moves going on simultaneously. Specifically, the new energy is calculated by each processor on the basis of histograms that might be changing during the computation and the (shared) energy is based on the latest processor's update of its value.

The parallel methods follow the same annealing schedule and convergence criteria as the serial algorithm discussed previously. Next we discuss the differences of the parallel annealing methods.

The first method, referred to as Method A, involves locking of nets and chips. Not only the pair of chips considered for exchange, but also all the nets that involve these two chips are locked; this results in all the chips that lie on the locked nets becoming effectively locked. With this complete locking when a move is accepted, the wire length and congestion are correctly calculated by each processor in parallel. This method, however, restricts the number of processors that can be used for a given problem size.

The amount of the locking overhead and the maximum number of processors that can be used depend on how the locking of the nets is applied. The following variants of Method A can be considered:

- A less restrictive approach, which would permit more processors to operate, is to lock only those chips that are connected by a single bond to each chip of the pair considered for exchange. In some cases the nets can be such that they connect only a pair of chips each; in this case this approach coincides with Method A.
- An equally less restrictive approach, which would also permit more processors to operate than in the strict case but is harder to implement than the previous method, is the following: For every two pairs of chips picked by the various processors, construct the two geometrical rectangles that contain each pair of chips. Then, even if the chips lie on the (same) locked nets but their bounding rectangles do not overlap, they can be considered for an interchange. We have not implemented this variant, as we opted for another method discussed later. In a recent paper Kravitz and Rutenbar [7] implement a form of this variation, but they appear to impose additional synchronization for the update of the energy when a successful move is found, not allowing chaotic computation to occur.

Method A and its variations yield more locking overhead and can in general employ fewer processors than the second method, discussed next.

In Method B only the chips considered for a move are locked. We introduce more chaos into the algorithm, but with this method there is less locking overhead and more processors can be used: In principle, as many processors as half the number of chips could be used.

In this method, when simultaneous moves are made by the various processors, the wire histograms are at times calculated incorrectly because of the relaxation on the net locks. However, the moves are based not on the absolute value of the energy but on differences between the *old* value and the *new* value that would result from the attempted

move. Though the differences themselves can also be incorrect, the error should be smaller. Thus we felt that such an approach might still be successful. We tried two variants of this approach.

Method B1 The correct histograms of the wires, and therefore the energy, are recalculated at each temperature, when the trials at this temperature are completed. The computation involved to perform this correction is negligible but processor synchronization is required (*barrier* [4]) at this point; because this is done once at each temperature, it does not cause performance degradation.

Method B2 To push this algorithm to its limits, we even ignore the correction of the histograms and energy at each temperature.

Variants of Methods B1 and B2 were considered but not implemented for the current model problem: In Method B1 one could restore the correctness of the histograms more frequently, that is, for every given number of histories at each temperature. Also, in Method B2, corrections could be made every few temperature steps.

Recent work by Casotto, Romeo, and Sangiovanni-Vincentelli [8] allows, as we do, errors in the computation of the energy but differs from our approach in that it partitions the chips among the processors and then (dynamically) moves chips from processor to processor to minimize the error in the energy.

We have implemented Method A and the variants B1 and B2 of the second method.

5. Virtual speedup measurements

The measurements in this study were performed in a parallel emulation environment, VM/EPEX [4] running on a dedicated IBM 3081 two-processor system. The EPEX system permits simulation of a shared-memory multiprocessor system with up to 64 processors. In EPEX, the multiple processors are emulated by multiple processes (Virtual Machines under the VM Operating System); we will hereafter use the terms (emulated) *processor* and *process* interchangeably. All processes have read/write access to user-specified segments of shared memory. Synchronization of these processes and communication among them are performed via semaphores in shared memory, using for example the *compare-and-swap* instruction, without operating system involvement. When more than two processes are used, the two physical processes time-share execution of the parallel program.

For each virtual machine executing, the VM Operating System measures the CPU time spent, referred to as the *virtual time*. Speedup measurements are deduced from the elapsed virtual time of the various processes applied to the problem. This is done as follows: Suppose N processes P_1, P_2, $\cdots$, P_N are applied in parallel to one program execution, resulting in elapsed virtual times T_1, T_2, $\cdots$. Suppose also that the program can be executed by one process using virtual time T. The "virtual speedup" is defined to be $VS = T/\max\{T_i\}$.

Because the simulated annealing algorithm is nondeterministic, virtual speedups are derived from timing measurements of multiple runs over independent problem samples. The estimated virtual speedup (VS_e) is the average time of a serial run divided by the average $\max\{T_i\}$. That is, $VS_e = \langle T \rangle / \langle \max\{T_i\} \rangle$.

The virtual speedup provides an approximation of the speedup which would be attained if the code were executed on a true N-way shared-memory multiprocessor such as RP3 [3]. The accuracy of this approach is limited by hardware and software overheads of multiprogramming that differ from those present in a true N-way parallel system. We have observed that the VM scheduler does not schedule all virtual machines in a *round-robin* manner but in a more irregular fashion, designed to optimize throughput in the entire system. However, in our parallel environment where all N virtual machines are applied to the execution of a program, the scheduler tends to give one virtual machine more time than the others even if the underlying program is entirely parallel. This makes our virtual speedups more pessimistic than they would be in true parallel hardware. The effect becomes more pronounced the more processes we apply; specifically, for more than 16 processes, we start seeing skews of the order of 20% (or larger) of the maximum virtual CPU time from the average time for each of the processes.

The VM scheduler provides time slices to the various running processes of duration several thousand instructions. Two processes cannot interact more frequently than that unless they are scheduled simultaneously on the two processors. The virtual speedup measurements do not accurately reflect the parallelism which can be obtained from assigning units of work less than a time slice, nor do they reflect *all* interactions that would occur in a true parallel machine.

Despite these limitations, the EPEX simulation environment captures the most important distinctions between the serial and the parallel algorithms. The parallel SA algorithms have two ingredients which would not occur in a serial algorithm: First, multiple processors use locks to ensure correct updating of the state; second, the calculation of the energy cost of a move does not take into account simultaneous moves which would alter the cost. Our measurements take these effects into account.

6. Performance measurements

In executing a problem in parallel, a measure of performance is how fast the job completes, i.e., the wall-clock speedup for that job. In addition, when the parallel algorithm is quite different from the serial, the quality of the parallel algorithm is gauged by examining how well it converges in comparison with the serial algorithm. In solving

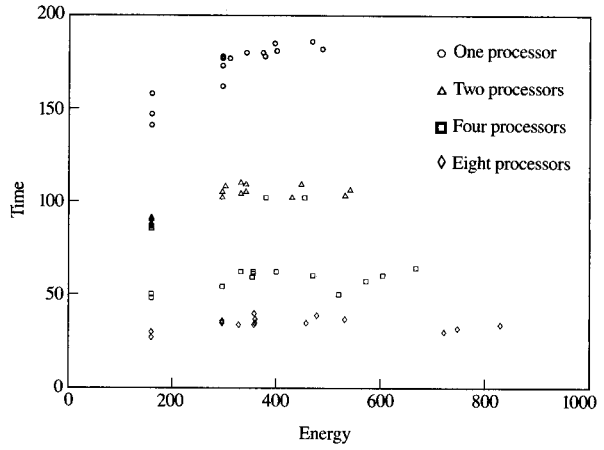

Figure 2

Time scatter plots for various final energies (from Method A).

Table 1 Average speedups.

	Number of processors					
	1	*2*	*4*	*8*	*16*	*32*
Speedup — Method A	1	1.7	3.0	5.9	7.1	—
" " B1	1	1.7	3.3	6.9	11.1	14.1
" " B2	1	1.9	3.6	6.6	9.5	11.7

deterministic problems, the speedup can usually be calculated by comparing the time required to execute the original (serial) algorithm vs. the time it takes to achieve the same result running on multiple processors.

In nondeterministic problems such as SA, the criteria above can be applied, but only in a statistical fashion. For example, we measure how well the parallel algorithm performs by the frequency of successful convergence compared with the frequency of convergence of the serial algorithm. In principle, speedups should be determined by comparing times needed to achieve the same result (energy) in serial and parallel execution, but in nondeterministic problems even the time needed to achieve the same result can vary. Another method we apply is to consider statistical speedups, that is, speedups averaged over multiple runs and various final energies.

• *Parallel SA algorithm performance measurements*

We have carried out the following performance measurements to examine the quality of the parallel algorithm:

- Parallelization overheads and speedup numbers between the serial execution and the parallel execution.
- Frequency of success and the distribution of the energies for the various algorithms.
- Annealing curves:
  a. Average energy as a function of temperature: $\langle E(T) \rangle$ vs. T.
  b. Final energy at each temperature as a function of temperature: $E(T)$ vs. T.
  c. Average value of $|\Delta E|$ for the the accepted moves as a function of temperature.
- Effect of locking: the overhead it incurs and the limitations it imposes in the parallelism.

• *Speedups*

As the algorithm is nondeterministic, we chose the following ways to display the speedup numbers. First, we display distributions of execution times for a number of runs as a function of the number of processors for each of the two parallel methods and the original serial algorithm. Second, we provide average speedup numbers; these were calculated by comparing computation times for 100 samples on varying the number of processors. A sample is generated by using a different initial configuration of the model problem and a different seed for the random number sequence. The computation time is the measured *virtual time* on System/370 [9], defined in Section 5.

The time distributions resulting from Method A are shown in **Figure 2** for various final energies and a varying number of processors; similar time distributions were obtained for the serial algorithm and the parallel method B. These plots show that the execution times are in general shorter when the problem converges, and that the execution times roughly tend to increase as the final energy increases.

By comparing the computation times for the serial algorithm and parallel methods A and B when run with one process, we determine the cost of inserting the parallelization routines [4] and the cost of imposing locks. On the average we find that the overhead of inserting the parallelization routines is about 15%. This parallelization overhead could be reduced by assigning, at a given temperature, more than one trial to each process each time; this is called *chunking* [4]. The *locking overhead* is due to the locks imposed on the chips and the nets. In Method B, where only the pair of chips considered for a move is locked, the overhead of locking is less than 1%. In Method A, where the chips and their nets are locked, the locking overhead is about 11%. The computation of imposing locks on the nets is more complex than simply imposing locks on the pair of the moved chips, and has the higher overhead.

The parallelization overhead remains roughly the same as the number of processors increases. The locking overhead, however, increases as the number of processors increases because of increased competition among the processors for

free chips. In **Table 1** we present speedups (VS_e, defined in Section 5) for each of the two methods. These speedups are average numbers over 100 runs for each set of processors. For each method, the speedups are normalized to 1 for one processor; in other words, we have included parallelization and locking overheads in the execution times of one process. The reason for doing that is to display more effectively the increase in the locking overhead as the number of processes increases.

As defined in Section 5, these speedups are measured by dividing the virtual CPU time for one process to execute the problem by the maximum virtual CPU time measured among the parallel processes. The time skew caused by the VM scheduler (Section 5) has the effect that the obtained virtual speedups for 32 processors for Methods B1 and B2 are lower than if, instead of the maximum time, we picked the average time spent by all (32) processes and defined the virtual speedup as $\overline{VS} = T/\langle T_i \rangle$. The estimated average speedup ($\overline{VS}_e$) can be defined analogously, and then the speedup for 32 processors for Method B1 becomes 16.7, and that for Method B2 becomes 15.9. For 16 processors, the corresponding numbers would be 11.4 and 10.8. Thus we conclude that in terms of their average speedups, both Methods B1 and B2 exhibit similar behavior.

In comparing the speedups for Method A and both Methods B we see that for two to four processors the speedup is nearly the same. As the number of processors increases, we see that the efficiency of Method A drops faster than that of either of the B methods; this trend persists also when we consider the average CPU time instead of the maximum CPU time to calculate the speedups. We believe that the reason for the difference between Methods A and B is increased contention due to the net locking.

For a fixed final energy, speedups can be calculated from only the runs that obtained that energy. Such speedups are shown in **Table 2** for obtaining the ground state (lowest energy).

It is intriguing to observe that Method B2 yields what could be considered *superlinear* speedups. Such superlinear behavior is also observed for higher values of the final energy. We find experimentally that with Method B2 the threshold of maximum number of trials at each temperature is reached at a much lower temperature than in the serial case and with Methods A and B1; thus, B2 performs fewer temperature steps involving the more expensive computation. However, we show in the following sections that B2 gives fewer good solutions, so it is not clear that the speedup shown in Table 2 can be used to advantage in a multiple-search procedure.

A correlation between the quality of solution and the speed with which it is obtained opens the possibility of efficient procedures in which several independent searches are conducted in parallel, all searches halting when one or two solutions have been found. Our work has not provided enough data to other than suggest this as an interesting possibility.

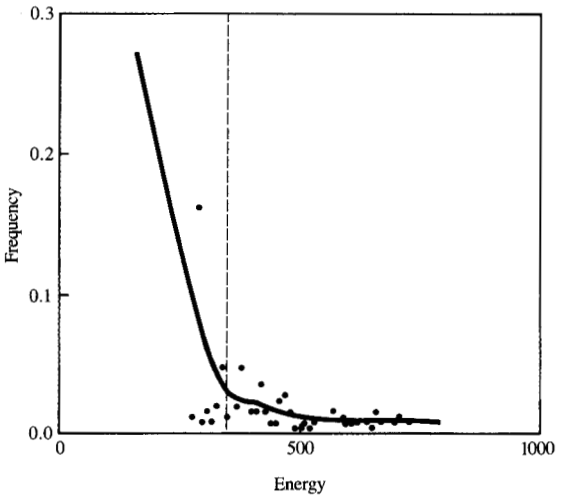

Figure 3

Frequency of final energies obtained with serial algorithm.

Table 2 Speedups in obtaining the ground state.

	Number of processors			
	1	*2*	*4*	*8*
Speedup — Method A	1	1.8	3.1	5.8
" " B1	1	1.8	3.4	6.4
" " B2	1	2.54	4.3	7.6

• *Distributions of energies*

Next we examine the distribution of energies over a sample of runs for the various algorithms. In **Figure 3**, we display the distribution of energies for the serial algorithm; **Figures 4(a)–4(b)** and **5(a)–5(b)** show the distributions of the energies for the parallel algorithms (Methods A and B1) for runs with two processors and eight processors. The solid line in these figures is a curve fitted through the points, and the vertical dotted line indicates the average energy obtained over the sample.

Energies below 300 correspond to final states which differ from the ground state by the transposition of a pair of chips or a slight *rotation* of some chips relative to their placement on the rectangular grid. In general, energies below 400 can be considered quite good; energies below 600 correspond to more complex rotations but may still be considered acceptable.

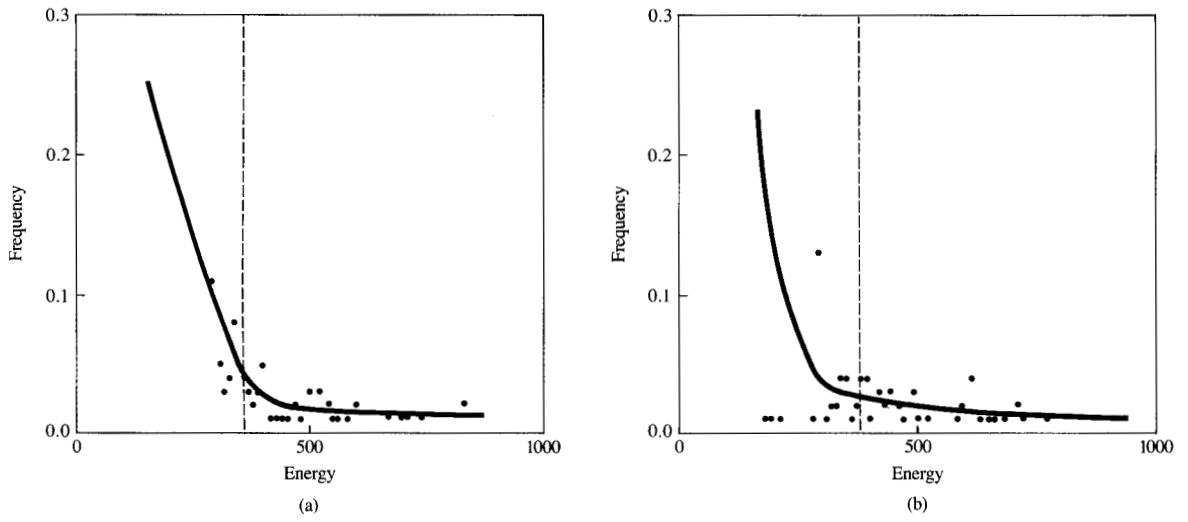

Figure 4

Frequency of final energies with parallel method A: (a) two-processor and (b) eight-processor runs.

We observe that both parallel algorithms find the correct result as frequently as the serial algorithm, and that the frequency of obtaining a given energy falls rapidly as the energy increases. The algorithm with net locks does as well as the serial algorithm for all the numbers of processors. It is encouraging to see that Method B1 also yields distributions of similar quality. Thus we conclude that Methods A and B1 behave as well as the serial algorithm.

Figures 6(a)–6(b) show distributions that were produced with Method B2. With this method too, for up to 16 processors, the frequency of obtaining the ground state is not low and the frequencies fall fairly rapidly for the higher final energies. We see, though, that the tails of the distributions get longer; that is, now we also find final energies that are considerably higher than with the other methods.

Annealing curves

To examine the quality of the parallel methods and attempt to explain their successes and failures, we examine the traditional measures of the simulated annealing algorithm, namely the annealing curves $E(T)$ vs. T and $\langle E(T)\rangle$ vs. T.

Figure 7 shows the $E(T)$ vs. T plot for an eight-processor run with Method A. Similar curves have been obtained for up to 16 processors with Method A and 32 processors with Method B1. We found no significant quantitative and qualitative differences from method to method, whether the problem converged or not.

A parameter examined often in SA [5] is the average value of the energy $\langle E\rangle$ as a function of the temperature. **Figure 8** shows a plot obtained for an eight-processor run with Method A. Corresponding plots obtained with the serial algorithm, for up to 16 processors with Method A and for up to 32 processors with Method B1, show similar behavior.

It should be emphasized that whether the problem has converged to the exact ground state or to an excited state, these annealing curves have similar shape, and they only change if the annealing schedule changes [1, 5]. Thus, the similarity of the plots of the parallel execution (Method A) and the serial execution indicates that the annealing schedule in the parallel algorithm is as good as the annealing schedule in the serial algorithm, despite the introduction of some chaos in the parallel methods.

Figure 9 shows the corresponding $\langle E(T)\rangle$ vs. T plot obtained for Method B2. In this method the energy of the system is calculated wrongly and can reach nonphysical (negative!) values of E at low temperatures. However, though the energy is calculated incorrectly during the annealing process, at the end, based on the final positions of the chips, the resulting configuration is legitimate.

Curves of the average fluctuations in the cost function

To examine the effect of chaotic energy calculation in Method B2, we examine another quantity: the average $|\Delta E|$ for the accepted moves; we call these the ΔE curves. The rationale behind examining such curves is that moves are decided on the basis of energy differences between an *old* value (before the move) and the *new* value (after the move). Small fluctuations in the average accepted $|\Delta E|$ at high and intermediate temperatures may indicate that the algorithm got stuck in a (high-lying) local minimum.

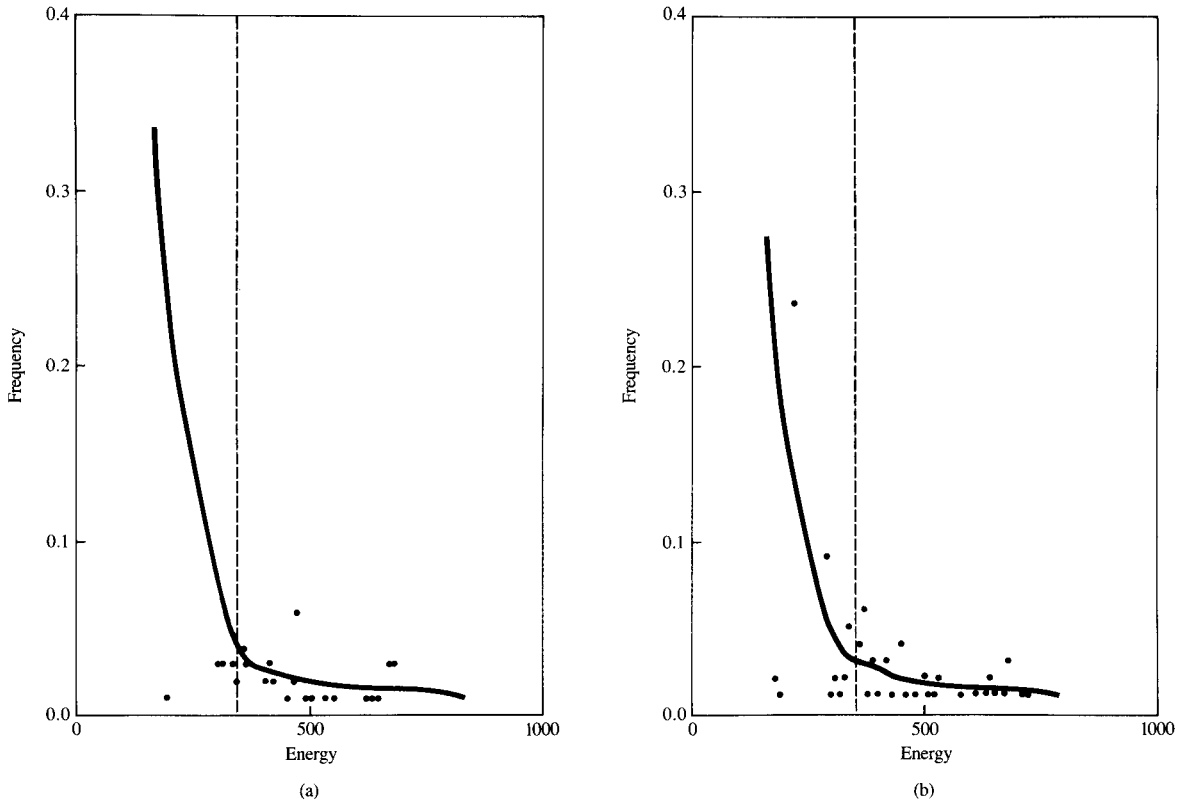

Figure 9

Frequency of final energies with parallel method B1: (a) two-processor and (b) eight-processor runs.

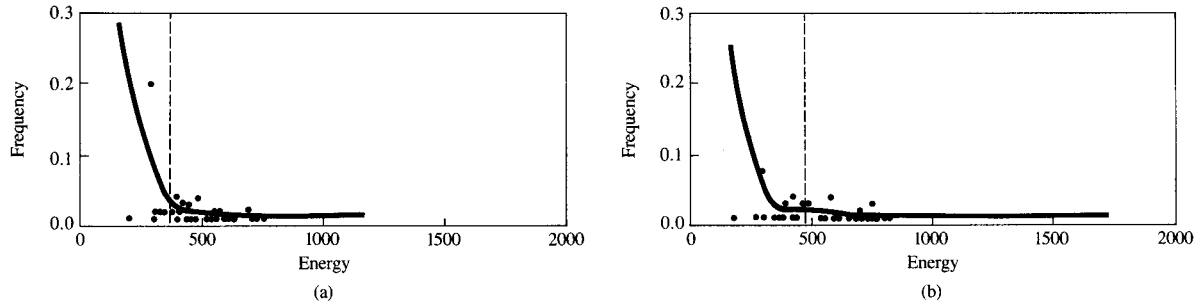

Figure 8

Frequency of final energies with parallel method B2: (a) two-processor and (b) eight-processor runs.

Such a plot is shown in **Figure 10** for the serial algorithm and parallel methods A and B1. **Figure 11** shows the corresponding plot for Method B2. Indeed, we see that curves of ΔE for Method B2, which has worse convergence

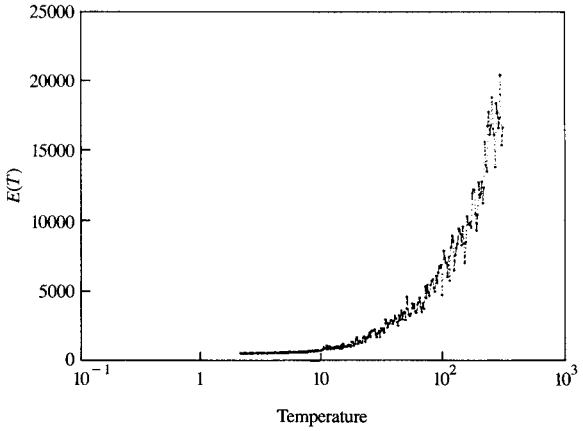

Figure 7

Typical $E(T)$ vs. T plot for serial algorithm and parallel methods A, B1, and B2.

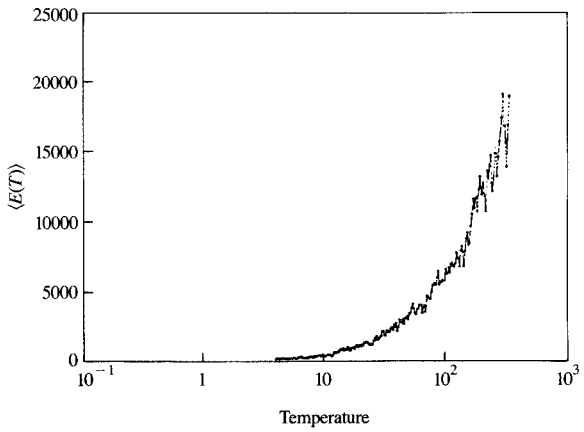

Figure 8

Typical $\langle E(T)\rangle$ vs. T plot for serial algorithm and parallel methods A and B1.

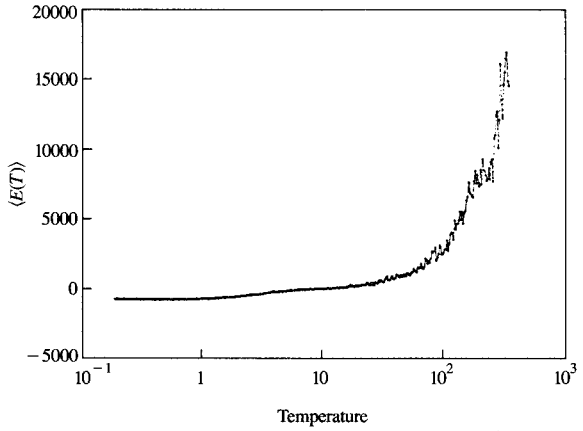

Figure 9

$\langle E(T)\rangle$ vs. T plot for Method B2.

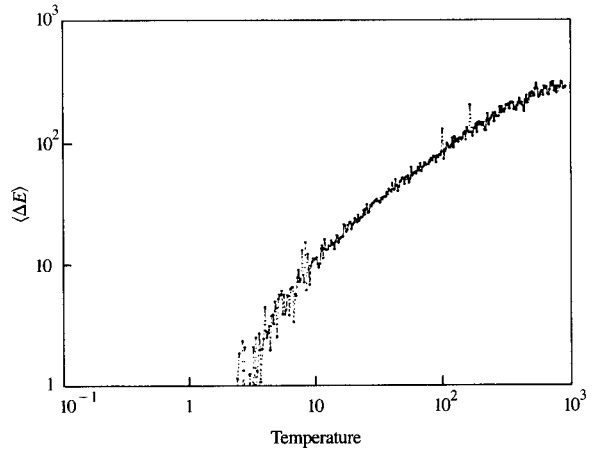

Figure 10

Typical $\langle|\Delta E|\rangle$ vs. T plot for serial algorithm and parallel methods A and B1.

features, are characteristically different from the corresponding curves for Methods A and B1, where we observe better convergence characteristics.

In general we observe that the ΔE plots of the well-behaved methods A and B1 show that at intermediate energies they allow larger fluctuations in energy, while the fluctuations in Method B2 are characteristically smaller. This behavior might cause the algorithm to become stuck at the high-lying minima, having less chance to escape to the lower-lying minima.

Thus, we have introduced an additional measure of the quality of the simulated annealing algorithm, namely the

average fluctuations in the energy (ΔE) for the accepted moves. The general usefulness of this quantity remains to be examined for other cases.

• *Locking*

In parallel SA methods such as the ones used here, where all processors work on the same chip configuration and locks are imposed on the chips, it is of interest to understand the overhead due to the locking, how this overhead increases as the number of processors increases, and what limitations the locking imposes on the maximum number of processors that can be employed.

From the elapsed virtual CPU time measurements we find that the static overhead of imposing locks on the chips is about 1% and of imposing locks on the nets is about 11%. Besides the longer execution time incurred by the net locking, another implication is the effect the net locking has on the maximum number of processors that can be employed for a given problem size. Following are our estimates on the limitations of the locking on the number of processors that can be effectively used.

Let n be the average number of chips locked each time a processor attempts to select a chip for exchange. In the case where we lock only the chips but not their nets, $n = 1$. In the case where not only a given chip but also its nets are locked, for our model problem, $n = 3.8$. Thus, each time a processor attempts to lock a pair, $2n$ chips are locked, that is, 2 chips and 7.6 chips for Methods B and A, respectively. This implies that, for our model problem ($N = 81$), the maximum number of processors is about 11 for Method A and 40 for Method B. In our experiments we have successfully made runs with up to 16 processors with Method A and with up to 32 processors with Method B. However, we find small improvement beyond 8 processors for Method A.

An additional loss in efficiency occurs because multiple processors compete to select pairs of chips, and thus the number of trials to select a pair increases. The average number of retrials as a function of the number of processors (P) is given by $N/[N - 2n(P - 1)]$, where N is the total number of chips in the problem and n is the number of chips locked every time a processor selects a chip for exchange. This estimate follows reasonably closely the average number of retries that we find experimentally as the number of processors increases.

7. Summary

We have studied two parallel approaches to the simulated annealing algorithm applied to the solution of the chip placement problem. Established criteria (annealing curves) have been examined and new ones (ΔE curves) proposed to investigate the quality of the parallel methods in terms of their convergence properties. Traditional parallel speedup performance measurements have been extended to statistical speedup measurements to handle the nondeterministic nature of the algorithm.

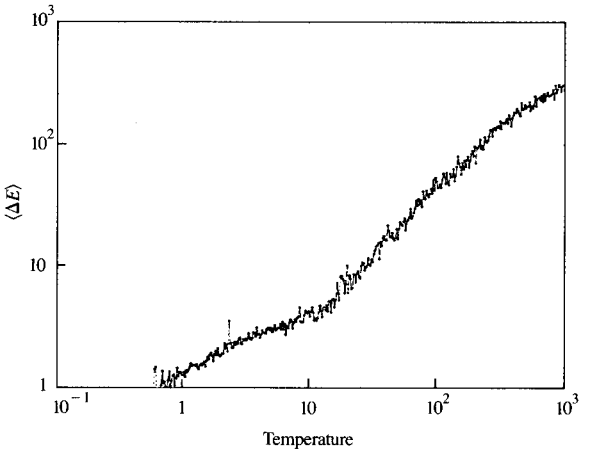

Figure 11

Typical $\langle |\Delta E| \rangle$ vs. T plot for parallel method B2.

The results we have obtained are encouraging. Of the approaches we have examined, we think that the most promising is Method B1, in which we lock only the chips, not the nets, and in which we resurrect the correct energy at the end of each temperature.

Our results indicate that with Method B1 we can employ a maximum number of processors that is of the order of 10–20% of the total number of chips, with efficiencies of about 80%. Considering that current realistic VLSI design problems involve the use of 5K–10K logic circuits, our results indicate that our proposed approaches can be used efficiently with highly parallel systems.

Acknowledgments

We thank Dr. Steve White, IBM Research Yorktown, for many helpful discussions.

References

1. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* **220**, 671 (1983).
2. P. J. M. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: A Review of the Theory and Applications*, to be published in July 1987 by Kluwer Academic Publishers, Boston, as part of their *Mathematics and Its Applications* series.
3. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Project (RP3): Introduction and Architecture," *Proceedings of the International Conference on Parallel Processing*, IEEE Computer Society Press, Los Angeles, CA, 1985, p. 764.
4. F. Darema-Rogers, D. A. George, V. A. Norton, and G. F. Pfister, "A VM Parallel Environment," *Research Report RC-11225*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY,

January 1985; J. M. Stone, F. Darema-Rogers, V. A. Norton, and G. F. Pfister, "Introduction to the VM/EPEX FORTRAN Preprocessor," *Research Report RC-11407*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, September 1985.
5. S. R. White, "Concepts of Scale in Simulated Annealing," *Proceedings of the International Conference on Computer Design*, IEEE Computer Society Press, 1984, p. 646.
6. R. H. J. Otten and L. P. P. P. van Ginneken, "Annealing: The Algorithm," *Research Report RC-10861*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, December 1984.
7. S. A. Kravitz and R. A. Rutenbar, "Multiprocessor-Based Placement by Simulated Annealing," *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, IEEE Computer Society Press, 1986, p. 567.
8. A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *Proceedings of the International Conference on Computer-Aided Design (ICCAD-86)*, IEEE Computer Society Press, 1986, p. 30.
9. *VM/SP Introduction*, IBM Program Product GC19-6200; available through IBM branch offices.

Received October 1, 1986; accepted for publication December 22, 1986

Frederica Darema *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Darema received a B.S. in mathematics from the University of Athens, Greece, in 1969, an M.S. from the Illinois Institute of Technology in 1972, and a Ph.D. from the University of California at Davis in 1976, both in theoretical physics. She was a research associate at the University of Pittsburgh and at Brookhaven National Laboratory, and a technical staff member at Schlumberger-Doll Research before joining IBM Research in 1982 as a Research staff member. Her research interests are in the areas of parallel algorithms, techniques, and tools for the development and performance analysis of parallel applications.

Scott Kirkpatrick *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Kirkpatrick received the A.B. degree from Princeton University, New Jersey, and the Ph.D. degree from Harvard University, Cambridge, Massachusetts, both in physics. He held a postdoctoral fellowship at the University of Chicago in 1970 and 1971, joining IBM in 1971. Since that time he has been at the Thomas J. Watson Research Center. His research has been principally in the physics of amorphous or disordered materials, such as spin glasses. This has led to work on design automation for VLSI and optimization by stimulated annealing. He is currently in the Computer Science Department at IBM Research. Dr. Kirkpatrick has been a visiting faculty member at the State University of New York, Stony Brook, and at the University of Paris.

V. Alan Norton *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Norton received the B.A. degree from the University of Utah, Salt Lake City, in 1968, and the Ph.D. from Princeton University, New Jersey, in 1976, both in mathematics. He was an instructor at the University of Utah (1976–79) and an assistant professor at Hamilton College (1979–80) before coming to IBM Research. Currently he is a Research staff member at the Thomas J. Watson Research Center, managing the parallel applications and architecture group of the Research Parallel Processing Project (RP3). His research interests include the performance analysis and architecture of parallel computer systems, parallel algorithms, fractals, and computer graphics.