Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCS-88-12

1988-04-01

Parallel Simulated Annealing

Roger D. Chamberlain, Mark N. Edelman, Mark A. Franklin, and Ellen E. Witte

Since the paper by Kirkpatrick, Gelatt and Vecchi in 1983, the use of Simulated Annealing (SA) in solving combinatoric optimization problems has increased substantially. The SA algorithm has been applied to difficult problems in the difficult problems in the digital design automation such as cell placement and wire routing. While these studies have yielded good or near optimum solutions, they have required very long computer execution times (hours and days). These long times, coupled with the recent availability of the number of commercial parallel processors, has prompted the search for parallel implementations of the SA algorithm. The goal ahs... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Chamberlain, Roger D.; Edelman, Mark N.; Franklin, Mark A.; and Witte, Ellen E., "Parallel Simulated Annealing" Report Number: WUCS-88-12 (1988). *All Computer Science and Engineering Research.* https://openscholarship.wustl.edu/cse_research/769

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/769

Parallel Simulated Annealing

Roger D. Chamberlain, Mark N. Edelman, Mark A. Franklin, and Ellen E. Witte

Complete Abstract:

Since the paper by Kirkpatrick, Gelatt and Vecchi in 1983, the use of Simulated Annealing (SA) in solving combinatoric optimization problems has increased substantially. The SA algorithm has been applied to difficult problems in the difficult problems in the digital design automation such as cell placement and wire routing. While these studies have yielded good or near optimum solutions, they have required very long computer execution times (hours and days). These long times, coupled with the recent availability of the number of commercial parallel processors, has prompted the search for parallel implementations of the SA algorithm. The goal ahs been to obtain algorithmic speedup through the exploitation of parallelism. This paper presents a method for mapping the SA algorithm onto a dynamically structured tree of processors. Such a tree of processors can be mapped onto both shared memory and message based styles of parallel processors. The parallel SA (PSA) algorithm is discussed and its performance evaluated using simulation techniques. An important property of the PSA algorithm presented is that it maintains the same move decision sequence as the Serial SA (SSA) algorithm this avoiding problems associated with move conflicts, erroneous move acceptance/rejection decisions and oscillations which have been associated with other PSA algorithm proposals. The PSA algorithm presented fully preserves the convergence properties of the SSA algorithm with speedups varying roughly as log2N where N is the number of processors in the parallel processor.

PARALLEL SIMULATED ANNEALING

**Roger D. Chamberlain, Mark N. Edelman,
Mark A. Franklin and Ellen E. Witte**

WUCS-88-12

April 1988

**Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899**

This research has been sponsored in part by funding from NSF under Grant DCR-8417709.

PARALLEL SIMULATED ANNEALING

by

Roger D. Chamberlain, Mark N. Edelman, Mark A. Franklin, and Ellen E. Witte

Computer and Communications Research Center
Washington University
St. Louis, Missouri

ABSTRACT

Since the paper by Kirkpatrick, Gelatt and Vecchi in 1983, the use of Simulated Annealing (SA) in solving combinatoric optimization problems has increased substantially. The SA algorithm has been applied to difficult problems in the digital design automation such as cell placement and wire routing. While these studies have yielded good or near optimum solutions, they have required very long computer execution times (hours and days). These long times, coupled with the recent availability of a number of commercial parallel processors, has prompted the search for parallel implementations of the SA algorithm. The goal has been to obtain algorithmic speedup through the exploitation of parallelism. This paper presents a method for mapping the SA algorithm onto a dynamically structured tree of processors. Such a tree of processors can be mapped onto both shared memory and message based styles of parallel processors. The parallel SA (PSA) algorithm is discussed and its performance evaluated using simulation techniques. An important property of the PSA algorithm presented is that it maintains the same move decision sequence as the Serial SA (SSA) algorithm thus avoiding problems associated with move conflicts, erroneous move acceptance/rejection decisions and oscillations which have been associated with other PSA algorithm proposals.. The PSA algorithm presented fully preserves the convergence properties of the SSA algorithm with speedups varying roughly as $\log_2 N$ where N is the number of processors in the parallel processor.

PARALLEL SIMULATED ANNEALING *

by

Roger D. Chamberlain, Mark N. Edelman, Mark A. Franklin, and Ellen E. Witte

Computer and Communications Research Center
Washington University
St. Louis, Missouri

1. INTRODUCTION

The use of Simulated Annealing (SA) in solving combinatoric optimization problems has increased substantially over the past five years. An important stimulus was the paper by Kirkpatrick, Gelatt and Vecchi [1] which showed how the original work by Metropolis et.al. [2] on simulating the annealing process could be applied to a broad class of combinatoric optimization problems. Of particular interest has been the application of SA to difficult problems (i.e., problems shown to be NP-complete) in the design automation of digital systems. Since then a host of research has been done on the application of SA to such problems as wire routing [3,4] and cell (circuit or chip) placement [5,6]. In addition, there have been studies on various aspects of the SA algorithm itself [7,8,9,10,11].

While these studies have demonstrated that the SA algorithm obtains good or near optimum solutions to a range of complex problems, they have also pointed out the need for very long computer execution times (hours and days) in achieving these solutions. This, combined with the recent availability of a number of commercial parallel processors [12,13,14,15], has prompted the search for parallel implementations of the SA algorithm [16,17,18,19,20,21], with the goal being to obtain algorithmic speedup comparable to the number of processors available in the parallel machine.

* This research has been sponsored in part by funding from the NSF under Grant DCR-8417709.

In this paper we present a method for mapping the SA algorithm onto a dynamically structured tree of processors. Such a tree of processors can be mapped onto both shared memory and message based styles of parallel processors. An important property of the Parallel SA (PSA) algorithm presented is that it maintains the same move decision sequence as the Serial SA (SSA) algorithm. This avoids problems associated with move conflicts, erroneous move acceptance/rejection decisions and oscillations which have been associated with other attempts at implementing PSA algorithms. Thus, the PSA presented here fully preserves the convergence properties of the SSA algorithm. The speedup obtainable with this algorithm varies roughly as $\log_2 N$ where N is the number of processors in the parallel processor.

The remainder of this paper consists of five sections. In the next section, the standard SSA algorithm is discussed and the PSA algorithm is presented and explained. In section 3 a simulation experiment is described. The experiment uses the travelling salesman problem as a test problem and is designed to determine the speedup (over a single processor) obtainable from the PSA algorithm with an N processor parallel computer. Section 4 presents and discusses the results of the simulation experiment. Section 5 concludes the paper and indicates how further increases in speedup over $\log_2 N$ are possible with variations on the PSA algorithm which use dynamically changing unbalanced trees of processors where the balancing is a function of the current algorithm position in the temperature schedule.

2. A PARALLEL SIMULATED ANNEALING ALGORITHM

2.1. The Serial Simulated Annealing Algorithm

The overall SA algorithm can be viewed in terms of a sequence of major cycles with each major cycle consisting of a series of N_{it} iterations. Each major cycle has associated with it a parameter referred to as the *temperature*. The temperature associated with the first (initial) cycle is T_i and the temperature of the final cycle T_f. From cycle to cycle the temperature changes according to $T_j = \alpha T_{j-1},\ 0.0 \le \alpha \le 1.0$. The algorithm terminates when the cycle for final

temperature is completed. Total number of major cycles completed is denoted as N_{mc}.

Each iteration within a major cycle is divided into three main phases (*move, evaluate* and *decide*). During the move phase a random change to the state of the system is performed. The evaluate phase recalculates the *energy* (or performance measure) of the system in its new state. The decide phase begins by calculating the difference between the old and new state energy values. If the energy has decreased then a decision is made to accept the move just performed (i.e., we now have a new system state). If the energy has increased then the move may be accepted, however, this is dependent on a statistically based decision. The distribution function involved in this decision has as a key parameter, the temperature, associated with the current major cycle. With the temperature "high," most of these increasing energy state changes will be accepted. With the temperature "low," most of these changes will be rejected. The temperature is initially started off at a high value with subsequent temperatures decreasing according to the parameter α until in the final major cycle the temperature is T_f. The overall algorithm is summarized in the pseudo-code given in Figure 1.

The algorithm, as given above, is essentially sequential in nature with each iteration within a major cycle dependent on the results of the prior iteration, and each major cycle dependent on the results of the prior major cycle. Maintaining the sequential nature of the algorithm is important since this is needed to guarantee convergence of the algorithm to the optimum or lowest energy state. A key aspect of the PSA algorithm given below is that the sequential properties of the SA algorithm are maintained while speedup through parallelism is achieved.

2.2. A Parallel Simulated Annealing Algorithm

At a given temperature, the SA algorithm can be viewed in terms selecting a particular path through a binary decision tree. Each iteration described above has associated with it a decision which results in either acceptance or rejection of a move (i.e., state change). Thus, the iteration process can be viewed as selecting a path in a decision tree which contains all possible

accept/reject decisions and associated paths for a major cycle. Figure 2, for example, shows a binary tree in which each of the nodes can be viewed as performing the move/evaluate/decide tasks with the resulting decision being selection of one of the two branches out of the node. A given path through this tree involves visiting three nodes and making three accept/reject decisions.

Let us now associate a processor with each of the nodes in the tree. Assume that communication paths exist between the processors so that paths along the node decision tree correspond to available communications paths between the processors. Assume that the processor to processor communications time is relatively small compared to the move/evaluate/decide tasks. Assume also that each processor has enough local memory to hold the state of the system and the processors operate in an MIMD mode. Various commercial hypercube computers roughly fit this parallel processor model.

Parallelism is achieved by recognizing several important features of the algorithm and the allocation of each move/evaluate/decide task to a different processor (See Figure 3).

1. Each processor along a reject path can start performing its move/evaluate/decide task as soon as its parent node has indicated start of the process (i.e., after a T_c delay). That is, since it is on a reject path no state change information must be communicated. This can be seen in Figure 3 in the start of the move task (T_m) on Node 2 relative to its parent on Node 1.

2. Each processor along an accept path can start performing its move/evaluate/decide task as soon as the move taken by its parent has been communicated. That is, since the new state of the parent has been communicated, a new move can be undertaken by the child without having to worry about inducing a state conflict. This can be seen in Figure 3 in the state of the move task on Node 5 relative to its parent on Node 1.

3. The computation time associated with evaluation (T_e) is a good deal larger than the the times associated with decision (T_d) and move (T_m). In addition, communications time is not a dominant factor.

Given these observations, the PSA algorithm begins by having successive nodes spawn tasks down the tree with each node then performing the move/evaluate/decide tasks. As indicated in Figure 3, many of these tasks can be performed in parallel. Only one path through the overall

tree, however, corresponds to the path finally selected. This final path selection begins when Node 1 finishes its decide phase. At this point it communicates its decision to its selected child (since this amounts to very little information it has not been shown on the diagram of Figure 3). When this child has completed its decision phase it, in turn, communicates this to its selected child. This goes on until a complete path through the tree is taken.

The dashed path on Figure 2, for example, indicates the accept/reject/accept path through the tree. This in turn corresponds to the linked nodes in the space-time diagram of Figure 3. In this diagram it is clear that with the seven available processors a speedup factor of about 2 is obtained. Notice that in the limiting case, as the evaluation time (T_e) grows, with other times becoming negligible, we effectively have all the evaluations along every path being performed in parallel. Thus, the limiting speedup achievable is $O(log_2 N)$.

At a given temperature, hundreds or thousands of iterations are necessary. This results in a decision tree and number of nodes far exceeding the number of available processors even for the largest current commercial parallel processors. This can be dealt with in several ways. The most straightforward approach is to have the final node in the selected path communicate its final state to the root node and then restart the process. In the above example, after Node 6 has completed, it would communicate its decision to Node 1 which would then continue the decision process. We refer to this as the STATIC PSA algorithm.

Another approach is to dynamically reassign nodes as they become available. Using the example above, after Node 1 has decided to accept its move, all of the nodes (processors) along the reject path out of Node 1 can be reassigned. That is, the left half of the tree can be pruned since our decision process has led us down the right half. If these nodes (and Node 1) are now reassigned as leaves in the tree, the tree can be continued. This is illustrated in Figure 4 which shows the reassignment of Nodes 1, 2, 3 and 4 (double circled) after the decision phase associated with Node 1 has taken place. The algorithm incorporating this process of dynamically reassigning

nodes (processors) we refer to as the DYNAMIC PSA algorithm. The limiting speedup with $T_e \rightarrow \infty$ is still $O(log_2 N)$, however, the actual speedup will be greater than the STATIC PSA case. This is due to the fact that in real situations communications, move, and decision times are not negligible and, with the DYNAMIC PSA algorithm, further evaluations can be performed in parallel with these times thus increasing the resultant overall speedup. In the STATIC PSA case this is not possible.

In the next two sections a performance analysis of these two PSA algorithms is presented. The performance analysis is based on a simulation experiment which utilizes the travelling salesman problem as a test case.

3. A PSA ALGORITHM SIMULATION

In this section the effectiveness of the PSA algorithms is determined by comparing their performance to the performance of the serial SA algorithm on identical problems. The execution time of the SSA algorithm, T_{ssa}, is easily computed given the number of major cycles (N_{mc}), the number of iterations per major cycle (N_{it}), and the SA time parameters (T_m, T_e, T_d).

$$T_{ssa} = N_{mc} * N_{it} * (T_m + T_e + T_d) \tag{1}$$

The execution time of the two PSA algorithms can be determined using a discrete event simulator to simulate each PSA algorithm on a given test problem. Because the PSA algorithms maintain the same move decision sequence as the serial algorithm, the simulator can be driven by the accept and reject decisions made by the serial algorithm. This ensures that decisions made by the SSA and PSA algorithms match. The test problem selected is the well known travelling salesman problem. In the remainder of this section we discuss this problem and the design of the discrete event simulator.

3.1. SA Applied to the Travelling Salesman Problem

The travelling salesman problem consists of a set of cities along with the distances between each pair of cities. The problem is to determine a trip (i.e., an ordered list of cities) such that each city is visited once and that the total distance travelled is minimized. The overall trip must be a cycle; that is, it must begin and end at the same city. The problem is combinatorially complex and has been shown to be NP complete.

To put this problem into the simulated annealing framework it is necessary to define a system state, an energy function and a move function. The system state is defined as a trip such that each city is visited once and the trip is a cycle. The energy function is chosen to be the distance travelled for the current system state (i.e., the sum of the distances between the cities visited on the trip). It is this performance measure which is to be minimized. The move function is a bit more complicated. The move function must be capable of generating a wide range of changes in the system state. There are a number of possible move functions which would fit this criteria. We chose pairwise-exchange as the move function. In pairwise-exchange, two trip positions are chosen at random and the cities in those positions are exchanged. For example, say that problem contains five cities. If the current system state (3 4 2 1 5) and the random position numbers 2 and 4 were chosen (the left most position is position number 1), then the new system state would be (3 1 2 4 5).

For the experiment discussed here, 40 cities were randomly placed within a unit square area. The SA algorithm began at a temperature $T_i = 50$. The temperature was decreased using $\alpha = .95$ until the temperature reached $T_f = .01$. At each temperature, 200 iterations were performed. This choice of initial and final temperatures dictates that the algorithm is performed over a wide range of move acceptance/rejection probabilities. When the temperature is high the probability of accepting a move is close to 1, while towards the end of the temperature cycle this probability approaches 0. The performance of the PSA algorithms will be seen to vary with the

acceptance probability.

3.2. The Design of a PSA Simulator

The performance of the PSA algorithms on a hypothetical machine can be modeled in a straightforward way by a discrete event simulator. The design of the simulator requires a set of assumptions about the hypothetical machine. The assumptions made are given in Section 2. To summarize, we assume an MIMD machine with communication paths between all processsors, local memory at each processor sufficient to hold the system state, and equal communications times associated with information interchange between processors.

The principal events which occur during the simulation correspond to the three phases of an iteration within a major cycle. These are: a) the completion of a move, b) the completion of an evaluate, and c) the completion of a decide. The PSA algorithm also requires the addition of events to handle system overhead. A communications event is necessary to model the passing of information from a parent processor to an accept or reject child processor. Note that the reject processor simply needs to know that it may begin a move/evaluate/decide task (it needn't know the rejected move since the system state hasn't changed). An accept child processor also needs to know the result of the move phase at the parent so that its system state can be updated. Although there is some difference in the amount of information which must be conveyed in each of these cases, for the purposes of simplicity they have been taken as being equal. Another overhead event is needed to model the pruning of the tree in the dynamic PSA case. This event corresponds to a processor being halted and reassigned to the bottom of the tree to perform a new move/evaluate/decide task. We generally assume that this takes little time compared to typical task evaluation times.

The simulator processes current events (by removing them from a time ordered event list) and schedules future events (by placing them on the event list) following a standard discrete event simulation algorithm as dictated by the logic of the PSA algorithm and the times associated

with the various tasks. For example, if the current event is the completion of the move task on processor 1, the simulator will schedule the completion of the evaluate task for processor 1. The completion of a move also indicates that an accept child processor can begin a move/evaluate/decide task. Therefore, the simulator will also attempt to allocate a processor to be the accept child and schedule the completion of a communications task (communicating the parent's move) at the allocated processor. The time of the scheduled events is determined by the current simulation time and the time to perform the scheduled activity.

A key issue in the design of the simulator relates to the handling of processor allocation. The specifics of processor allocation are determined by the type of PSA we want to simulate (i.e., static or dynamic). In the static PSA case the processors are initially allocated to form a binary tree with fixed parent/child links. When either the root or an internal node is ready to communicate with a child node, the child node is easily determined since processors are allocated to fixed tree positions. Parent nodes communicate with their children to start execution of a move/evaluate/decide task. This goes on throughout the tree, the timing being determined by the times associated with the various tasks. The leaf node processors simply process their own move/evaluate/decide tasks without triggering tasks at other processors. One leaf node, however, corresponds to the selected decision path from the root through the tree. When this leaf node has finished its decide phase it communicates its state to the root node and the process begins again. This continues through each major cycle until the major cycle associated with the final temperature has completed.

In the dynamic PSA case, the situation is more complex. An idle processor pool is maintained by the simulator and this pool is accessed to allocate child processors as needed. Initially the pool contains all of the processors except one. This one is the root of a tree which grows and is pruned dynamically. Certain paths from the root will be ready to grow faster than other paths. Specifically, the path of all reject children will grow fastest because a reject child can be triggered as soon as the parent is triggered (the move of the parent need not be calculated).

On the other hand, the accept child cannot be triggered until the parent has completed the move phase. Due to this time difference between accept and reject paths, nodes in the reject paths may use up the entire pool of processors. To avoid this we control tree growth so that each level in the tree is completely filled before growth continues to lower levels. This ensures that all paths contain the same number of processors before any single path is extended. Because the tree is kept balanced, we refer to this as dynamic BALANCED PSA. This method assumes that each path is equally likely, therefore there is no reason to grow any path more quickly than any other path. In fact, the assumption that all paths are equally likely is incorrect. By utilizing information about the likelihood of each path we can extend these ideas to develop a dynamic UNBALANCED PSA algorithm in which the tree is allowed to grow in an unbalanced manner. The unbalanced growth can be controlled by knowledge about the likelihood of a path being the selected path. In this way we can achieve even higher levels of parallelism. Further study of the dynamic UNBALANCED PSA algorithm is a topic of ongoing research.

The discrete event simulator implemented can simulate PSA algorithm performance independent of the particular application on which the algorithm is used. The time parameters of the application and the serial decision sequence associated with the SSA algorithm operating on the application are necessary inputs to the simulator. The simulator accesses the serial decision sequence when the current event is the completion of a decide task. The serial decision determines the result of the decide task and, based on the result and the type of PSA algorithm being simulated, the simulator schedules future events. For example, the completion of a decision in the dynamic PSA algorithm requires the pruning of one side of the tree. Which side is pruned is determined by the result of the decision. If the decision is to accept, the reject side of the tree is pruned and vice versa. The simulator schedules events to indicate that the pruned processors should halt execution and rejoin the pool of available processors.

4. PSA ALGORITHM PERFORMANCE

The performance measure of interest is the speedup of the PSA algorithms over the SSA algorithm. This speedup depends upon the the number of processors in the PSA implementation (N), the relative values of the time parameters (T_m,T_e,T_d,T_c), and the probability that a move is accepted or rejected. The time parameters, with the exception of T_c, are dependent on the application and on the effectiveness of program implementation of the various tasks. In this paper we examine the speedup associated with four different sets of time parameter values and number of processors. We also examine how the speedup varies with temperature during execution of the PSA algorithm.

Speedup is defined as the ratio of the execution time for the SSA algorithm to the execution time for the PSA algorithm. SSA algorithm execution time was given earlier in equation 1. The execution time of the PSA algorithms is determined by running the simulator until $N_{mc} * I_{it}$ decisions have been made and the major cycle associated with the final temperature has completed.

In the experiment presented here, the PSA algorithms were simulated with the number of processors set at N=3,7,15,31. Four different time parameter sets were considered. To simplify time parameter specification several assumptions were made about their relative values. Specifically, we assumed $T_c = T_d$, $T_m = 2 * T_d$, and $T_e = K * T_m$. K is an independent variable used to produce different time parameter ratios.

These assumptions are based upon experience with application of the SA algorithm to the travelling salesman problem. Given the difference in energy between the current state and the proposed next state, the decide phase takes a constant, short amount of time. This time is independent of application. The time for the move phase is dependent on application, but is often relatively short since the move simply permutates the current state. The evaluate is also application dependent and will vary significantly from one application to the next. By varing K

(K=2,4,8,16) we can take into account this application dependence. For the travelling salesman problem K is about 2.

Figure 5 is a graph of the speedup attained by the static PSA algorithm as the number of processors and the time parameter ratios were varied. The curve corresponding to K infinite is theoretical and is the maximum speedup possible with this algorithm for the given number of processors. There are two trends of interest in this graph. First, as the number of processors increases the speedup increases. When there are more processors, the decision tree is deeper. This means that there are more move/evaluate/decide tasks underway along each path from the root. Second, as the evaluate phase takes longer compared to the move and decide phases (K increases) the speedup increases. When K is small, the evaluate phase is comparable to the move and decide phases. This means that a given node finishes its decision phase before very many of its descendants have had tasks spawned. The node remains idle until the leaf corresponding to the selected path finishes its decide phase, the leaf communicates to the root to begin a new task and the spawning of children reaches the given node. When K is large, the evaluate phase is long compared to the move and decide phases. Because the evaluate phase is long, when a given node finishes its decision phase many, if not all, of its descendants have had tasks spawned. The time the node will be idle will be decreased since significant progress will have been made on each path from the root to the leaves.

The theoretical maximum speedup occurs when K is infinite. In this case the evaluate takes so long that the communicate, move and decide phases essentially take no time. This means that the tree is immediately filled with tasks at every processor. When the root finishes execution every other processor has finished execution also. Only one path from the root is the selected path, and this path contains $\log_2 N$ processors. Thus, there are $\log_2 N$ decisions made in the time the serial algorithm could make one decision.

Figure 6 is a graph of the speedup attained by the dynamic PSA algorithm as the number of processors and the time parameter ratios were varied. The curve corresponding to K infinite is theoretical and is the maximum speedup possible. The same two trends discussed for the static PSA occur in the dynamic PSA. First, as the number of processors increases the speedup increases. When there are more processors, the likelihood increases that an allocation request can be filled rather than be placed on the waiting list. As fewer requests are required to wait, the algorithm can execute more quickly. Second, as the evaluate phase takes longer compared to the move and decide phases (K increases) the speedup increases. When K is small, the root will finish its decide phase before the tree has grown very deep. There will be processors which are idle because not many requests for child processors will have been made. When K is large, the tree will have time to grow. If K is large enough, the tree can grow to use all the available processors and place some allocation requests on the waiting list. The speedup increases because the paths from the root have had time to grow longer.

As in the static PSA case, the maximum speedup for the dynamic PSA occurs when K is infinite. In this case the tree grows immediately to use all the available processors. When the root finishes execution every other processor has finished execution also. The selected path contains $\log_2 N$ processors, thus the speedup is $\log_2 N$.

It is interesting to compare the performance of the two PSA algorithms. For all values of K the dynamic algorithm is faster than the static algorithm. This is because the dynamic algorithm makes better use of idle processors. In the dynamic algorithm, idle processors are added to the pool of available processors. In addition, the dynamic algorithm prunes the tree when a decision is made to halt execution at processors along the path which is not selected. The pruned processors are also added to the pool of available processors. The static algorithm requires that processors be idle until the selected path has been completed and the root can once again initiate the spawning of processors from the top of the tree. As K increases, the difference in performance between the dynamic and static algorithms decreases. When K is large, the dynamic

algorithm allocates all the processors and is forced to wait for the completion of the decision phase at the root. With K large the static algorithm also allocates all the processors and must wait for the same reason. In this case the pruning and reallocation of processors in the dynamic case does not take place and therefore the performance of the two algorithms is similar.

Figure 7 is a graph of the speedup attained by the dynamic PSA algorithm for the case $K = 2$ as the temperature changes. The speedup versus temperature is given for 15 and 31 processors. As expected the speedup is greater for 31 processors than for 15 over the entire range of temperatures. It is interesting to observe that the speedup varies significantly over the range of temperatures. For 31 processors the speedup is less than 2.5 for a temperature of 50 and approximately 4.5 for temperatures less than 1. As the temperature changes the probability a move is accepted changes. Because the reject children can be spawned before the accept children, a sequence containing all reject decisions will execute more quickly than a sequence containing some accept decisions. When the temperature is high the decision sequence will contain primarily accept decisions. When the temperature is lowered the decision sequence contains increasingly more and more reject decisions. At each temperature a fixed number of decisions are made. When the temperature is lowered this fixed number of decisions will execute more quickly than at higher temperatures because of the higher number of reject decisions. When the temperature is less than 1 almost all decisions are reject decisions so the speedup approaches a constant.

5. SUMMARY AND CONCLUSIONS

This paper has presented and determined the performance of a pair of new parallel simulated annealing algorithms. By maintaining the same decision sequences associated with the standard serial SA algorithm, the parallel SA algorithm maintains same convergence properties associated with the serial algorithm.

The algorithm is based on mapping the decision tree associated with the serial algorithm onto a tree of processors which operate in parallel/pipelined manner. Operating in this way, many paths through the decision tree are evaluated simultaneously with a single path eventually being selected. The maximum speedup achievable with these algorithms is $\log_2 N$, with the actual speedup dependent on the relative times associated with the move, evaluate and decide subtasks, and on the number of processors available. With the static parallel SA algorithm there is fixed mapping of the decision tree onto the available processors. With the dynamic parallel SA algorithm, idle processors are kept available in a common pool, and assigned as needed in evaluating selected branches of the tree. The result is that the dynamic algorithm achieves higher speedup then the static algorithm.

An interesting extension to these parallel algorithms would take advantage of the predominance of accept decisions when the temperature is high, and the predominance of reject decisions when the temperature is low. Knowing the probability of the accept/reject decision, one could construct unbalanced trees of processors which would better match the expected decision probabilities. In this situation, at high and low temperatures one could expect speedups approaching N, with the $\log_2 N$ bound coming into play only during those temperatures when the probabilities of reject and accept are equal. Speedups much greater the $\log_2 N$ should be possible. This varient on the above algorithms is now being studied.

REFERENCES

1. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science* **220**(4598) pp. 671-680 (May 13, 1983).

2. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *J. Chem. Phys.* **21**(6) pp. 1087-1092 (June 1953).

3. M.P. Vecchi and S.A. Kirkpatrick, ''Global wiring by simulated annealing,'' *IEEE Trans. on Computer-Aided Design* **CAD-2**(4) pp. 215-222 (Oct. 1983).

4. H.W. Leong, D.F. Wong, and C.L. Liu, ''A simulated-annealing channel router,'' *Proc. Int. Conf. on Computer-Aided Design*, pp. 226-228 (Nov. 1985).

5. C. Sechen and A. Samgiovanni Vincentelli, ''The Timberwolf placement and routing package,'' *IEEE J. Solid-State Circuits* **SC-20**(2) pp. 510-522 (April 1985).

6. L.K. Grover, ''A New Simulated Annealing Algorithm for Standard Cell Placement,'' *Proc. Int. Conf. on Computer-Aided Design*, pp. 378-380 (Nov. 1986).

7. S.R. White, ''Concepts of Scale in Simulated Annealing,'' *Proc. Int. Coonf. Computer-Aided Design (ICCD84)*, (Dec. 1985).

8. D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, ''Convergence and Finite-Time Behavious of Simulated Annealing,'' *Proc 1985 Decision and Control Conf.* , (Dec. 1985).

9. M.D. Huang, F. Romeo, and A. Samgiovanni Vincentelli, ''An Efficient General Cooling Schedule for Simulated Annealing,'' *Proc. Int. Conf. Computer-Aided Design (ICCAD86)*, pp. 381-384 (Nov. 1986).

10. J.W. Greene and K.J. Supowit, ''Simulated Annealing without Rejected Moves,'' *Proc. Int. Conf. on Computer-Aided Design (ICCAD84)*, pp. 658-663 (Oct. 1984).

11. F. Romeo and A. Sangiovanni-Vincentelli, ''Probabilistic hill climbing algorithms: Properties and applications,'' *Proc. Chapel Hill Conf. on VLSI*, pp. 393-417 (1985).

12. Intel Scientific Computers, ''iPSC: The first family of concurrent supercomputers,'' *INTEL Product Announcement*, (1985).

13. Sequent Computer Systems, ''Balance 8000: Guide to parallel programming,'' *Sequent Product Announcement*, (July 1985).

14. NCUBE Computer Systems, *The NCUBE Parallel Processor*

15. Bolt Beranek and Newman Inc., ''Butterfly 1000,'' *BBN Product Announcement*, (1987).

16. Andrea Casotto, Fabio Romeo, and Alberto Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *IEEE Trans. on Computer-Aided Design* **CAD-6**(5) pp. 838-847 (September 1987).

17. Saul A. Kravitz and Rob Rutenbar, "Placement by Simulated Annealing on a Multiprocessor," *IEEE Trans. on Computer-Aided Design* **CAD-6**(4) pp. 534-549 (July 1987).

18. F. Darema, S. Kirkpatrick, and V.A. Norton, "Parallel Techniques for Chip Placement by Simulated Annealing," *Proc. Int. Conf. on Computer Design (ICCD87),* pp. 87-90 (Oct. 1987).

19. C.P. RaviKumar and L.M. Patnaik, "Parallel Placement by Simulated Annealing," *Proc. Int. Conf. on Computer Design (ICCD87),* pp. 91-94 (Oct. 1987).

20. M.H. Jones and P. Banerjee, "An Improved Simulated Annealing Algorithm For Standard Cell Placement," *Proc. Int. Conf. on Computer Design (ICCD87),* pp. 83-86 (Oct. 1987).

21. P. Banerjee and M. Jones, "A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer," *Proc. Int. Conf. on Computer-Aided Design (ICCAD86),* (Nov. 1986).

$T \leftarrow T_I$
while $(T \geq T_f)$
  for $i = 1$ to N_{it}
    move
    evaluate
    decide
  endfor
  $T \leftarrow \alpha T$
endwhile

Figure 1: The Serial Simulated Annealing Algorithm

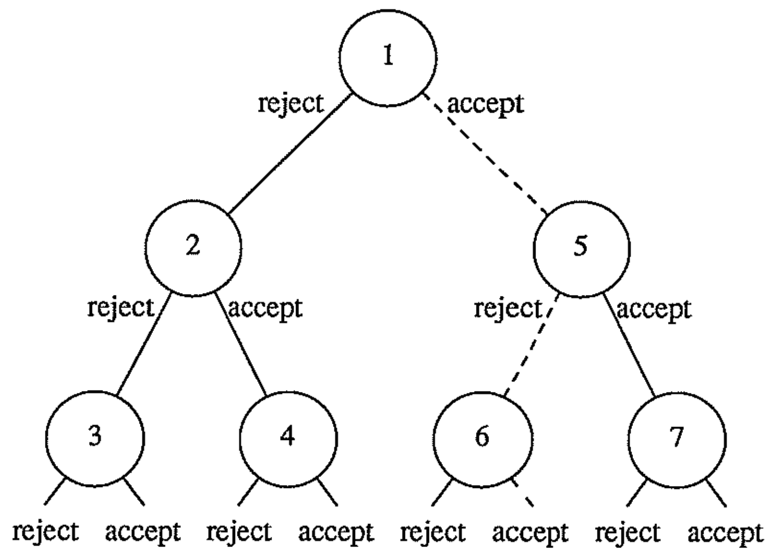

Figure 2. SA Decision Tree

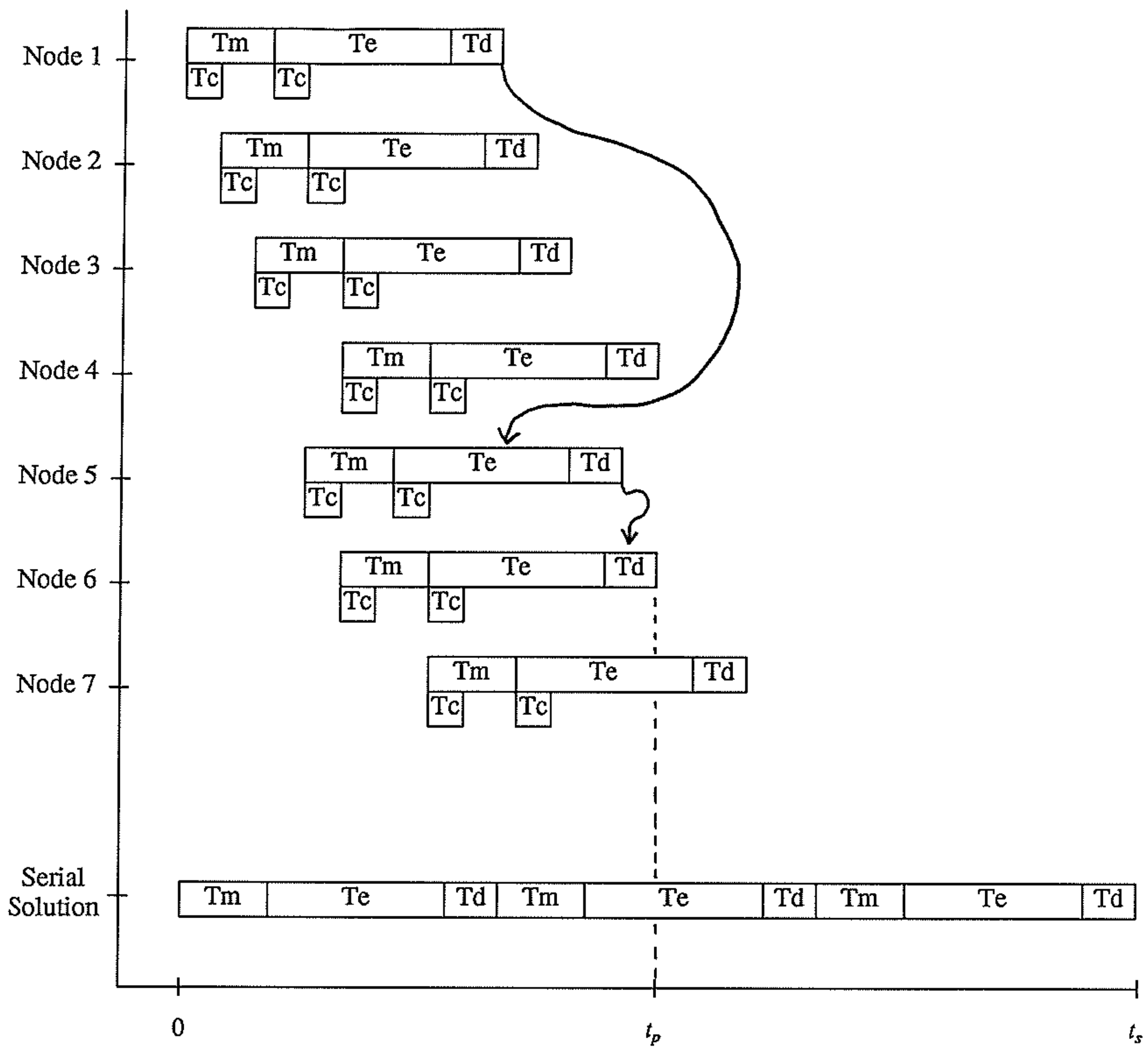

Figure 3. PSA Timing Diagram

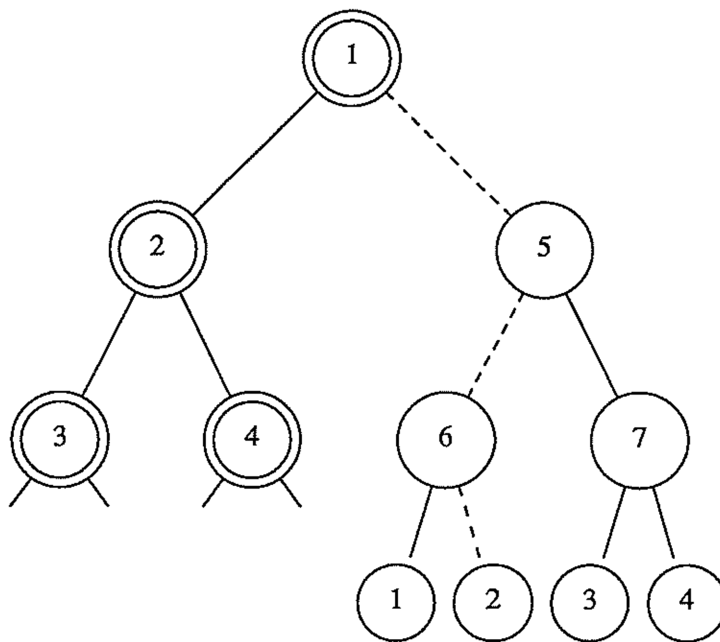

Figure 4. Node Reassignment in Dynamic PSA Algorithm

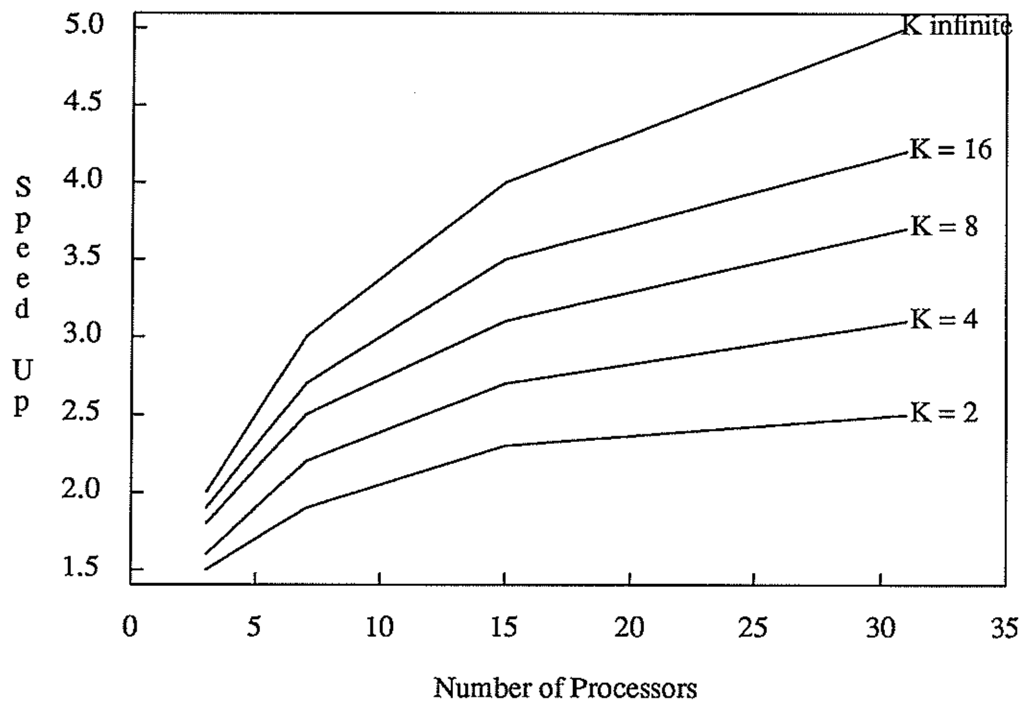

Figure 5. Speedup v. Number of Processors for Static PSA

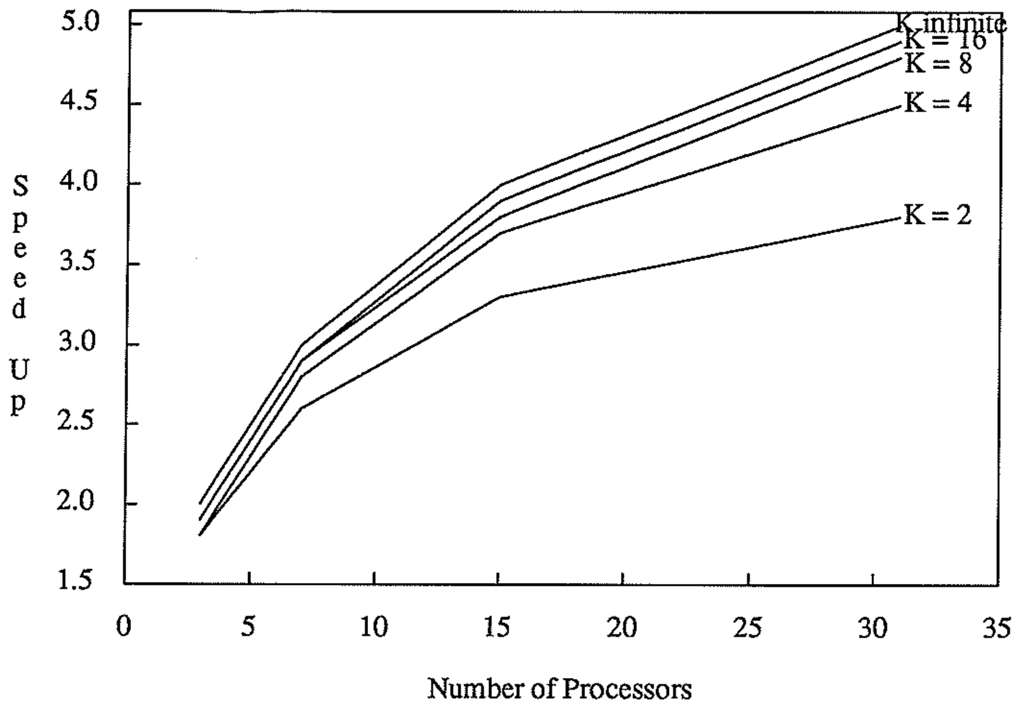

Figure 6. Speedup v. Number of Processors for Dynamic PSA

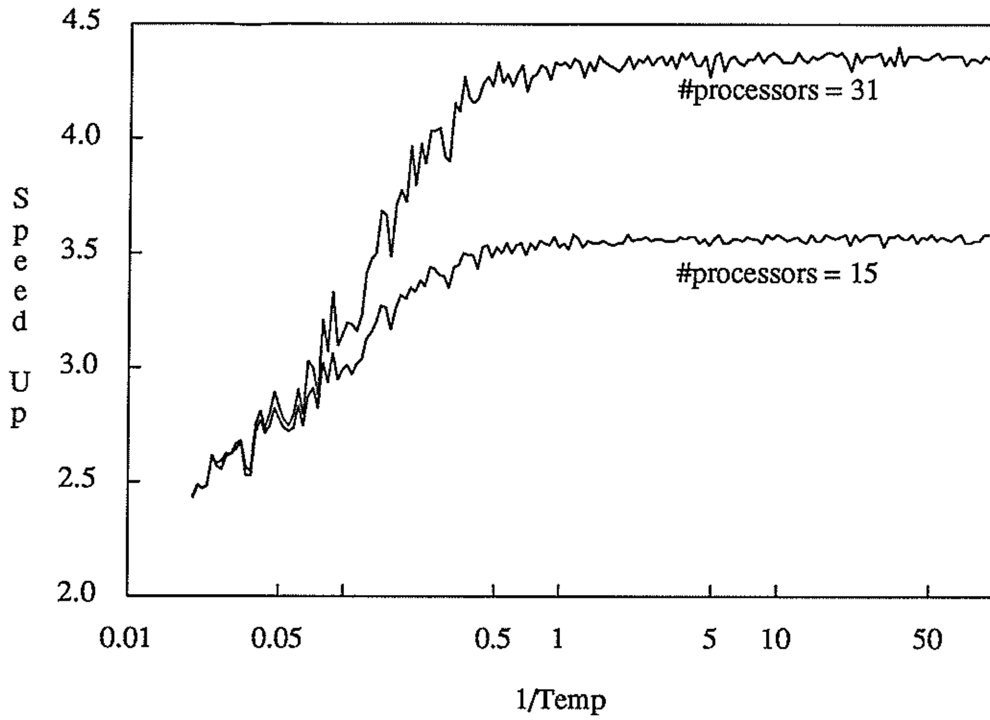

Figure 7. Speedup v. 1/Temperature for Dynamic PSA (K=2)