AIAA JOURNAL
Vol. 33, No. 12, December 1995

Parallel Processing in Optimal Structural Design Using Simulated Annealing

Mir M. Atiqullah* and S. S. Rao†
Purdue University, West Lafayette, Indiana 47907-1288

The need to reliably solve large structural design optimization problems in a reasonable time frame naturally leads to the investigation of the scope of parallel processing in optimization. Parallel optimization may be approached in two general ways. First, the analysis could be the target of parallel processing specially if it is significantly large compared with the overall computation. The second approach is the parallel implementation of the optimization algorithm, which can then be easily adapted for solving any appropriately formulated optimization problem. In this work the basics of parallel computer architectures and some popular parallelization strategies are described. Simulated annealing (SA), a stochastic, discrete optimization technique, is chosen for its global capability, robustness, and suitability for parallel processing. The concept of "shakeup" in SA, which simulates re-annealing, is introduced. The beneficial effects of shakeup for escaping local optima is demonstrated by solving a two-dimensional multimodal optimization problem. A pin-jointed 10-storied multibay plane truss structure is considered as an example optimization problem and has been solved using both serial as well as parallel versions of the SA algorithm. The parallel result, achieved with a relatively small parallel configuration of the computer, indicates that very large structural designs can be optimized in much shorter times along with high probability of achieving global solutions even on moderately sized parallel computers.

I. Introduction

WITH the advent of advanced computers, engineers kept pace by coming up with problems requiring even higher computational speed and power to solve larger and more complex structural and other design problems. As the speeds of computers approach their limiting values, the manufacturers introduced various innovations like pipelining, vector processors, etc., to further improve the throughputs. Although these vector processors or so called "supercomputers" are again pushed to their physical limits, the need for higher speeds and lower costs is driving the current trend towards development of parallel computers. Parallel computers are simply multiple processing units combined together in an organized fashion such that multiple independent computations for the same problem could be performed simultaneously or concurrently, thereby further increasing the overall computational speed.

Several parallel computing architectures have been developed in recent years.[1–3] Because of multiplicity of parallel architectures, many investigations explored their topology, structures, functionality, and applicability for solving computationally intensive problems. Multiprocessor connectivity, an important characteristic feature, is the topology of the physical interconnection between the individual processing units and how they communicate with each other. The connectivity affects, and often defines, the capability, characteristics, and limitations of a specific parallel processor system. It also defines the adaptability of a parallel system for implementation of a specific approach of parallelization. Siegel[4] presented a study of various interconnection networks for large-scale parallel processing.

The first approach, of parallelizing the analysis, has been used by many researchers. Among these, structural analysis and design, which are computationally quite challenging, have been frequently studied. The finite element method is widely used in structural analysis, which fortunately lends quite well to the "divide and compute" strategy of parallel processing. These situations have led to numerous research papers on the parallel implementation of the finite element method for solving structural design and optimization problems.[5–7]

Although the concept of annealing was known for some time, it was formally introduced to the arena of optimization only recently.[8] There have been a number of research studies reported[9] to further formalize simulated annealing (SA) and to improve its effectiveness as a robust optimization tool. A brief literature review indicates that little research has been done on parallel design optimization and even less on global optimization techniques. Dixon and Patel[10] and Dixon et al.[11] have done some research on the utilization of parallel processing in design optimization. Because of its combinatorial and global capabilities, SA enjoyed tremendous popularity in the areas of complex discrete optimization such as cell placement in microprocessors and routing design on printed circuit boards. On the other hand, due to its computationally intensive nature, SA could not gain as much popularity in the solution of problems involving numerous variables such as those encountered in structural optimization.

With the rapid development of massively parallel computers and research on parallelization techniques, SA could augment its effectiveness for structural optimization tasks. Roussel–Ragot and Dreyfus[12] described the aspects of scaling and parallel implementation of simulated annealing. Some publications have appeared on the utilization of iPSC/860 systems in implementing parallel processes, including a LINPACK benchmark established on an intel Touchstone Delta iPSC/860 system.[13] A large number of publications have appeared on SA and the utilization of parallel processors in the late 1980s and early 1990s. As larger and efficient parallel computers emerge, parallel SA may become more competitive with traditional optimization methods. Moreover, there have been some algorithmic developments of SA in terms of its convergence,[14,15] which provided SA with an additional competitive edge in terms of computational time and the quality of results.

A brief review of the basic simulated annealing algorithm is helpful in setting a stage for explaining our development of the algorithm and its implementation. This review is given in Sec. II along with a description of the proposed convergence improvement strategy dubbed "shakeup." A multimodal unconstrained optimization problem is used to demonstrate the effectiveness of the shakeup strategy in escaping local minima. In Sec. III we describe the various factors that separate one parallel processing system from another. The common methods of achieving parallelism in structural design as well as the logic behind parallelizing the SA algorithm are given. Several performance measures of parallel computer systems are

Received April 12, 1994; revision received Feb. 8, 1995; accepted for publication Feb. 9, 1995.

*Graduate Teaching Assistant, School of Mechanical Engineering.
†Professor, School of Mechanical Engineering.

described in Sec. IV. Expressions are developed for computing these performances of a parallel application running on a hypercube (one of the ways processors are connected) system. Section V provides a simple theoretical development on the performance criteria for parallel processors, specifically for processors using hypercube interconnection topology. Section VI provides a representative structural optimization problem with moderate complexity to demonstrate the effectiveness and the computational efficiency of the parallel SA algorithm. Conclusions are drawn in Sec. VII along with some direction towards future research.

II. Optimization Using Simulated Annealing

A. Constrained Optimization and Simulated Annealing

All structural design tasks may be stated as optimization problems where one or multiple design objectives are minimized (or maximized), subject to various material, functional, and other design constraints. Design variables may include the shapes and sizes of the individual elements of the structure. The design problem can be stated as follows:

Find a vector of design variables

$$\bar{X}$$

that minimizes (maximizes) an objective function

$$f(\bar{X}) \tag{1}$$

subject to

$$\Phi_m(\bar{X}) \leq 0; \qquad m = 1, 2, \ldots, M$$

and

$$\Psi_n(\bar{X}) = 0; \qquad n = 1, 2, \ldots, N$$

where M and N are the number of inequality and equality constraints, respectively. During optimization, repeated structural analysis, using finite element method, is performed to assess the response of the structure and subsequently adjust the design variables to minimize (maximize) the defined objective.

Simulated annealing (SA) is a combinatorial optimization technique originally developed for the optimization of problems with integer and discrete variables. The name is derived from the simulation of thermal annealing of critically heated solids. Slow and controlled cooling of such solids ensures proper solidification with minimum possible level of internal energy. The Monte Carlo method has been used to simulate such statistical randomness in change of state of a design. This process randomly changes the variables to a new set of neighborhood values such that a certain objective is improved over many such steps. Since SA works using only objective evaluations, a constrained design objective must be reformulated taking the constraints into consideration. Using a penalty function approach to construct an unconstrained objective $F(\bar{X})$ is a straightforward and widely used technique, as shown in Eq. (2):

$$F(\bar{X}) = f(\bar{X}) + \sum_{m=1}^{M} a_m \Phi_m(\bar{X}) + \sum_{n=1}^{N} b_n \, abs[\Psi_n(\bar{X})] \tag{2}$$

where a_m and b_n are suitable positive scaling factors for inequality and equality constraints, respectively. The basic SA works using the following two steps in one cycle of its operation:

Step 1: Randomly perturb the variables and calculate the resulting change in objective function (ΔF).

Step 2: If $\Delta F \leq 0$, accept the design unconditionally; else if $\Delta F > 0$, accept the design with a probability

$$p = \exp\left[\frac{\Delta F}{k^* T}\right]$$

else reject the design and reverse the perturbation of step 1.

Here T is the control parameter dubbed "temperature," which is decreased in a controlled fashion as the annealing progresses. The parameter k is a suitable scaling factor (simulating the Boltzmann constant for solids) that affects the acceptance criteria for occasional worse designs. The iterative process of SA, including several stochastic processes, is shown by a simple flow diagram in Fig. 1. The temperature decrement scheme, dubbed "cooling schedule," plays a very sensitive role in the convergence characteristics of simulated annealing. Various cooling schedules were proposed in the literature.[23] A simple but effective cooling schedule is implemented in this work such that when current temperature T_k is reduced, the statistical acceptance probability in step 2 is also reduced by a constant fraction α, on transition to the next temperature T_{k+1}:

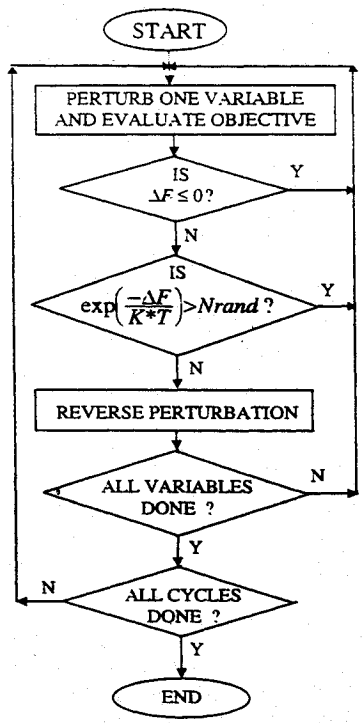

Fig. 1 Flow diagram of basic simulated annealing algorithm.

$$T_{k+1} = \alpha T_k \tag{3}$$

where $\alpha = [T_f / T_s]^{1/n}$, T_s is the starting temperature (usually 0.5–0.9), T_f is finishing temperature, usually $\mathcal{O}(10^{-8})$, and n is the maximum number of steps allowed for decreasing the temperature.

Since SA is a combinatorial optimization method, it must be specially adapted for handling continuous variables encountered in a vast majority of engineering problems including some structural designs. An exponentially decreasing stepping criterion is proposed such that after sufficient number of stages of annealing, the apparently discrete step size S_k of each transition will become very small and virtually continuous as

$$S_{k+1} = \beta S_k \tag{4}$$

where the decrement factor, $\beta = \exp\{\ell_n(S_f/S_s)/n\}$, S_s is the starting step size, and S_f is the finishing step size (on the order of the accuracy desired).

At the beginning the temperature T is kept high enough to probabilistically accept a majority of worse designs. This criterion is unique in SA, which allows it to probabilistically jump across local optima. As the iterative process continues with decrease in temperature T, it becomes increasingly unlikely that any worse designs will be accepted and at the final stages it behaves much like a greedy iterative updating scheme. Given numerous transitions to allow the design to settle at each intermediate temperature and as T approaches zero, the SA algorithm will asymptotically arrive at the global optimum. Since the number of such transactions needed are infinitely many, solutions obtained after the practically feasible number of transactions can be claimed as asymptotically global at best. Some of the notable features of simulated annealing are as follows:

1) The quality of the final solution is not affected by the initial guesses, except that the computational effort may increase with worse starting designs.

2) Because of the discrete nature of the objective function and constraint evaluations, the convergence or transition characteristics are not affected by the continuity and/or differentiability of the functions.

3) The functioning of SA is unaffected by the convexity status of the feasible design space.

4) SA can be adapted to handle mixed integer, discrete, or continuous variables.

5) The variables need not be positive.

B. Shakeup Strategy

The global capability of SA is asymptotic and can only be guaranteed by a homogeneous algorithm where annealing is performed for an infinitely large number of cycles and the temperature T is decremented in infinitesimally small steps. Neither of these conditions or the nonhomogeneous implementation of the SA algorithm is realizable in real life, and as such the global solution cannot be guaranteed. Convergence towards the solution depends on, among many others, the cooling schedule employed, proximity of the initial starting point to the solution, and the complexity of the domain. While negotiating a multimodal domain, a nonhomogeneous SA algorithm may very well be trapped in a local minimum. Assuming that the initial starting solution is a fair one and is not separated from the global solution by more than a few local solutions, a strategy dubbed shakeup is introduced to jump over these local minima. The purpose of shakeup is to decrease the overall computations needed for achieving the global solution in a multimodal space. The working of the shakeup strategy can be summarized by the following steps:

1) Set the maximum number of allowable shakeups. Set this number to a fraction (one-half, one-quarter, etc.) of the usual number of iterations and perform annealing.

2) Reset the annealing parameters and restart the procedure with the last obtained solution as the starting point and perform annealing.

3) Compare the current solution with the previous one. If the solution is improved, go to the previous step for another round of shakeup. Else stop.

The working of shakeup with relation to basic SA is shown by a flow diagram in Fig. 2. The probability that shakeup will find a solution better than the current (and presumably not the best) one is the same as that of the simple SA algorithm finding any minimum. Although it may appear that by increasing the number of annealing iterations one could achieve the results that the shakeup is supposed to achieve, it is advantageous in that by periodic revival of the probabilistic acceptance criterion and the longer jumps, it has a higher probability to jump over local hills, where it might have been trapped. This capability is demonstrated by finding the global minimum of a two–dimensional function that has 15 stationary points. This function, a form of the well known "camelback" function, is given by

$$f = 4x^2 - 2.1x^4 + (x^6/3) + xy + 4y^4 - 4y^2 \qquad (5)$$

The 15 stationary points of this function are given in Table 1. The contour plot of this test function is shown in Fig. 3 along with the starting point $x = -2$, $y = -1$, $f = 5.7333$ (same for implementation with and without shakeup), and the final solutions. The shakeup augmented algorithm found one of the two global minima in the second round, whereas the simple algorithm got trapped near one of the local minima on its search path. Both algorithms ran for a total of 500 iterations. It can be seen that the simple SA algorithm found the solution $x^* = -1.73$, $y^* = -0.18$, and $f^* = 2.29$ (near point 11 in Table 1), whereas the SA with shakeup found the solution $x^* = -0.087$, $y^* = 0.72$, and $f^* = -1.03$ (vicinity of point 9 in Table 1). The convergences of the two algorithms against the number of iterations are shown in Fig. 4. Note that the shakeup is initiated at the midpoint (250 iterations) where the augmented algorithm temporarily accepted worse designs and got out of the local minimum and found a better solution. In computationally intensive applications such as structural design optimization, the shakeup strategy may prove to be an acceptable compromise between a local solution with limited computation and the global solution with prohibitively large amount of computation.

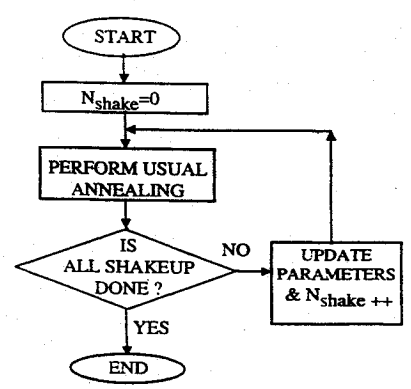

Fig. 2 Flow diagram showing the working of shakeup strategy in simulated annealing.

Table 1 Stationary points of the nonconvex camelback function

No.	x	y	$f(x, y)$
1	0.0	0.0	0.0
2	0.0898	−0.7127	−1.0316[a]
3	−1.1092	0.76827	0.5437
4	1.2302	0.1623	2.4963
5	1.2961	0.6051	2.2295
6	1.67071	0.5687	2.1043
7	1.6381	0.2287	2.2294
8	1.7036	−0.7961	−0.2155
9	−0.0898	0.7127	−1.0316[a]
10	−1.1092	0.76827	0.5437
11	−1.2302	−0.1623	2.4963
12	−1.2961	−0.6051	2.2295
13	−1.67071	−0.5687	2.1043
14	−1.6381	−0.2287	2.2294
15	−1.7036	0.7961	−0.2155

[a]Global optimum.

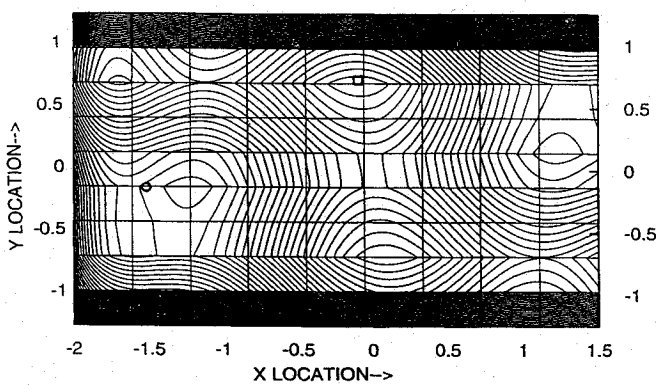

Fig. 3 Contour plot of the camelback function. The rectangular and the circular spots indicate the solutions achieved by SA with and without shakeup, respectively.

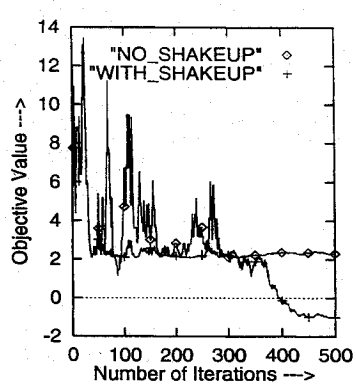

Fig. 4 Convergence of SA with and without shakeup for optimizing the camelback function.

III. Parallel Simulated Annealing and Structural Design

In large structural systems, the analyses are very much computationally intensive and somewhat intractable. Many researchers attempted to alleviate such problems by using different innovative techniques such as substructuring methods and multilevel decomposition of the problem.[16–18] Advantage has been taken of various computing architectures such as distributed processors and vector computers to further improve the computational efficiency. Although gradient-based methods have traditionally been used as optimization tools, many novel techniques, such as genetic algorithms (GA) and SA with advanced capabilities, have been recently used with promising results. This paper provides a methodology for structural optimization using a parallel implementation of simulated annealing on a multiple instruction multiple data (MIMD) parallel processor system with hypercube interconnection.

A. Parallel Computer Architectures

The differences between the various parallel computers are determined by processor–memory configuration and the number of independent instruction streams possible. Processor–memory relationship in a parallel computer could be either distributed memory or shared memory type. In a shared memory system, the total block of memory is accessible to all of the processors, whereas the distributed memory architecture provides a smaller but private memory for each processor. In terms of independent streams of instruction, the parallel processors may roughly be classified as either single instruction multiple data (SIMD) or MIMD. SIMD systems typically consist of a multitude of identical processing elements (PE), a large shared memory, and a small amount of processor memory. A centralized control unit (CU) connects all of the PEs through an interconnection network. The control unit broadcasts the same instruction to all of the PEs, which then execute the instruction on the data each PE has. Data may be transferred among the PE through the interconnection network. On the other hand MIMD computers are capable of running independent, probably related, programs simultaneously. Although program execution in SIMD machines is synchronized by virtue of architecture, the logic and data synchronization in MIMD must be programmed. A MIMD machine may be configured to use either distributed or shared memory, whereas SIMD machines are usually restricted to the later.

The topology of interconnection between the processors is another important criterion of how the parallel system will perform. Some popular topologies include linear array, mesh array, tree array, hypercube, and fully connected arrays. The parallel processor used for this research is an Intel iPSC/860 and uses a hypercube topology, which relies on direct communication channels between neighboring processors in a virtual multidimensional space. Each processing unit, called a node, is capable of direct communication with its logical neighbors. The size configuration n of a hypercube system must be n multiples of 2. For example, a three-dimensional hypercube architecture ($n = 3$) resembles a familiar cube with (2^3) 8 corners, each having a processing node.

B. Parallel Processing in Structural Design

For parallel processing of any problem it must lend itself to suitable decomposition without compromising the integrity of the original problem or accuracy of the analysis. Often the traditional way of computation needs to be changed significantly to be able to compute the components concurrently. In structural design, parallelism may be explored in one or more of the following six areas: 1) substructuring of the total structures, 2) solution of linear equations, 3) mathematical optimization technique, 4) modal analysis (if any), 5) dynamic analysis (if any), and 6) postprocessing (if any). One must be careful about one subtle aspect of parallel processing, namely, the interprocessor communication overhead. A divide and conquer strategy to compute in parallel will need a communicate and combine process that entails data communication among the processors that should be kept at a minimum. A large parallel computer can be desirable for dividing the problem into numerous smaller subproblems but will experience larger overhead due to increased communication needs. These aspects dictate the compromise between the number of processors utilized and the corresponding communication overhead to maximize the benefits of parallel processing.

C. Decomposition Strategies for Parallel Processing

The decomposition methods applicable to an arbitrary algorithm may be broadly classified under the following three categories: 1) perfectly parallel decomposition, 2) domain decomposition, and 3) control decomposition. For most of the application programs, any of the preceding methodologies would usually be suitable, but for a very large and complex system of application programs, a mixed strategy may be the choice. Some applications, such as computer graphics and image processing, lend themselves naturally to perfectly parallel decomposition techniques. These applications can be divided into parallel processes that need little or no communication and usually run for the same length of time on all processors, leading to a good load balance. This is the most obvious type of decomposition.

The domain decomposition is particularly suitable for three kinds of computations: 1) when data are static, 2) when certain data are tied to a single physical entity and may be assigned to a node, and 3) where the physical domain is fixed, but the data may be dynamic within the domain. The domain decomposition of the simulated annealing algorithm falls into the second category and is suitable for implementation on MIMD computers.

The control decomposition is the most complex strategy to implement and is suitable for problems where the aforementioned methods do not work well. Another control decomposition technique called "manager/worker " uses one processor as the manager (may be the host machine), which distributes and controls the tasks given to workers (nodes). The task assignment may very well be dynamic on a worker availability basis.

For very large applications, no single decomposition method may be suitable for the entire application. Rather, the application can be viewed as a set of layers, each of which may be decomposed using any of the preceding strategies. This is referred to as "layered decomposition " and is the most difficult one to implement.

D. Implementation of Parallel Simulated Annealing Algorithm

A practical aspect of domain decomposition is the availability of nodes on the iPSC/860 as compared with the number of design variables involved. To reduce the number of variables in an analysis, often several related variables are grouped into one. Although computationally favorable, this compromises the quality of true optimization. If grouping is not done, as in the case of the example structure (described later), the available nodes on a parallel computer can often be largely outnumbered by the variables to be manipulated. In this case the variables are systematically assigned (mapped) to the nodes such that the assignment is as even as possible. After mapping is planned, effective internode communication must be programmed to exchange data as and when needed across the nodes. The parallel implementation of the SA algorithm, along with the internode communication (global operations), is shown in Fig. 5.

Since the number of variables involved is much larger than the number of nodes available on the iPSC/860, a simple mapping technique (assigning few specific parameters to a specific node) was employed. Essentially each processor node runs its own annealing algorithm concurrently with others. The domain decomposition is achieved through dividing the variables among the available processors. Although each node proceeds with its own annealing, it manipulates only those variables assigned to it by the decomposition strategy. The global synchronization of all nodes is done followed by a global assembly of updated results from all other nodes at the end of each cycle. This ensures the availability of correct and updated data at all of the nodes for evaluation and computation in the next iteration. Although SA is inherently a discrete variable algorithm, the example problem is assumed to have continuous variables to demonstrate its adaptability to other types of variables. A strategy of diminishing step size in each subsequent cycle is adopted to simulate the continuous variables. This may result in a premature slowdown of the algorithm near later iterations, although considerable advantage is taken of larger step sizes at the beginning. This

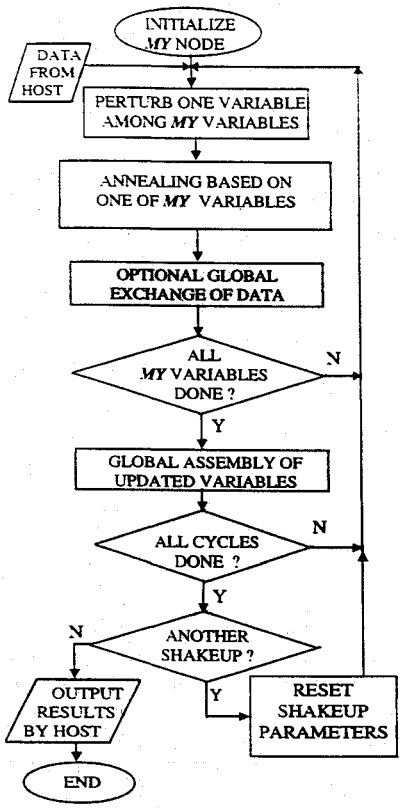

Fig. 5 Flow diagram of parallel SA on one node of the iPSC/860 parallel computer.

may be compared with an annealing process where a premature cooling restricts perfect crystallization and may require re-annealing or shakeup, as described in Sec. II.B.

In the present implementation, a node takes design decisions based on its share of parameters while it has no information regarding the decisions being taken concurrently by other nodes until the global data exchange at the end of the current cycle of annealing. This temporary lack of information may be viewed as part of the stochastic process and may result in temporarily imperfect decisions on the part of the individual nodes. The availability of global updating information can be improved, to some extent, at the cost of more internode communication time. As shown by an optional step in the flow diagram of Fig. 5, this may be implemented in the parallel algorithm to share, among all nodes, the updated variables as soon as they become available. The advantage of increased availability of information is that more informed decisions could be taken by individual processors, possibly improving the convergence. The disadvantage is that it will result in (N_{add}) additional sets of communications calculated as (rounded up to the next integer)

$$N_{\text{add}} = \frac{N_{\text{var}}}{N_{\text{node}}}$$

where N_{var} is the number of design variables and N_{node} is the number of nodes used. When a large number of iterations are involved, such as in SA, few additional communications in each iteration would make a significant difference in the overall solution time. This additional communication has not been implemented in the current implementation.

IV. Performance of Parallel Processors

As expected, the time taken by a parallel computer will be less than that of a traditional serial computer for a parallelizable computation. But the time gain is far from a linear relation with the number of processors used. Two most common terms used to measure and compare performances of parallel computers are the speed-up and the efficiency. The speed-up factor (S_p) on n nodes is defined as the ratio of time taken by a serial run to that of a parallel run of the same algorithm:

$$S_p = T_1/T_p \tag{6}$$

where T_1 is the time needed by a sequential algorithm to solve the same design problem and T_p is the time taken by the parallel algorithm. Although speed-up is a raw measure of the time gain, the incremental advantage of using an increased number of processors is also of interest. A term called efficiency (E_p) is used to account for the effect of the number of processors in the performance computation. It is defined as the speed-up per node,

$$E_p = S_p/n = T_1/(n \cdot T_p) \tag{7}$$

This expression gives a good estimate of the performance but is only good for near perfectly decomposed data because it ignores the communication factor. The expected performance criterion completely depends on the memory–processor relation, interconnection architecture, and speed of individual processors and communication links, as well as how the problem is parallelized or decomposed. Any computation, no matter how well it is decomposed, requires a portion of it that must be computed in a serial fashion. If the computational time in series (on one node) is denoted by T_1, it can be expressed as

$$T_1 = T_s + T_{p1} \tag{8}$$

where T_s is the time spent performing the serial part of computation and T_{p1} is the computation time spent on the part of the program that can be divided for parallel processing. If the program is run by dividing the parallelizable task into the number of processors n, then the total execution time in parallel T_p can be expressed as

$$T_p = T_s + T_{pn} + T_{\text{com}} \tag{9}$$

where T_{com} is the time needed for data communication and T_{pn} is the time for parallel execution of the parallelizable code; T_{pn} can be found by taking the quantity $(T_{p1})/n$. If T_{c1} is the average time needed for a single communication to the nearest neighbor, including switching and data transmission, the time needed for a global communication (most demanding) can be expressed as $d \cdot T_{c1}$, where d is the dimension of the parallel processor system. For a hypercube system with n ($= 2^d$) nodes, it can be expressed as $d = \{\ell_v(n)/\ell_v(2)\}$; therefore communication time is given by

$$T_{\text{com}} = \left\{\frac{\ell_v(n)}{\ell_v(2)}\right\} \cdot T_{c1} \tag{10}$$

If the number of global communications needed in the optimization algorithm is denoted by n_{com}, then the communication time T_{com} can be found by

$$T_{\text{com}} = \left\{\frac{\ell_v(n)}{\ell_v(2)}\right\} \cdot T_{c1} \cdot n_{\text{com}} \tag{11}$$

Substituting Eq. (11) for T_{com} in Eq. (9), we get

$$T_p = T_s + \frac{T_{p1}}{n} + n_{\text{com}} T_{c1} \cdot \left\{\frac{\ell_v(n)}{\ell_v(2)}\right\} \tag{12}$$

Equations (6), (8), and (12) yield

$$S_p = \frac{T_s + T_{p1}}{T_s + (T_{p1}/n) + n_{\text{com}} \cdot T_{c1} \cdot \{\ell_v(n)/\ell_v(2)\}} \tag{13}$$

Thus the efficiency, Eq. (7), can be expressed as

$$E_p = \frac{T_s + T_{p1}}{n(T_s + (T_{p1}/n) + n_{\text{com}} \cdot T_{c1} \cdot \{\ell_v(n)/\ell_v(2)\})} \tag{14}$$

The speed-up and efficiency relations, Eqs. (13) and (14), are specific to the hypercube architecture and applicable to the domain decomposed simulated annealing algorithm but are independent of the design problem being solved. Similar expressions for estimating the

performance of parallel processing systems utilizing various communication architectures can be developed. From Eq. (14), it can be seen that although the use of a larger number of nodes reduces the term T_{p1}/n, it also reduces the value of E_p. If the number of nodes utilized is beyond the saturation number, the efficiency would actually decrease with further increase in the number of nodes.

V. Numerical Example

A 4-bay, 10-storied pin-jointed plane structure was used as the example. The structure, shown in Fig. 6, is 100 in. high, 40 in. wide, has 53 nodes, and 128 members and is subjected to 2 loading conditions. In the first loading condition, a load of 200 lb is applied at node 49 toward right and a load of 400 lb is applied at nodes 2, 9, 14, 19, 24, 29, 34, 39, and 44 toward right. In the second load condition, a load of 15,000 lb is applied downward at nodes 49 and 51. The structure is fixed at the bottom nodes 1, 4, and 7. The structural design is to be optimized for minimum weight. There are constraints on each truss member due to maximum allowable stress (40,000 psia), buckling under axial compression, upper size limit (5.0 in.^{2}), and lower size limit (0.1 in.^{2}). Results obtained by solving this design problem by both serial and parallel computers are described next.

A. Serial Solution

The continuous variable version of the SA algorithm was implemented in Fortran 77 on a dual processor Gould NP1 sequential computer and tested for its effectiveness. Since continuous variables are simulated by programming monotonically decreasing step sizes, the progression is slow during later iterations. The shakeup strategy is employed to recover from such slowdown of convergence as demonstrated by the example in Sec. II. B. Figure 7 shows the convergence plots by solving the optimization problem on the NP1 and a Sun Sparc Station IPC. The results from NP1 indicate a surprisingly efficient convergence during the second shakeup. A possible explanation for this behavior is that first stage of shakeup converged somewhat prematurely and in the next stage the solution followed a steep valley and achieved a better result quickly. The convergence of the results from the Sun IPC was more typical of a multishakeup optimization. In both the cases, the weight was minimized from an initial 72.3 to the optimum of 21.6 lb, and no constraints were violated at the optimum solution.

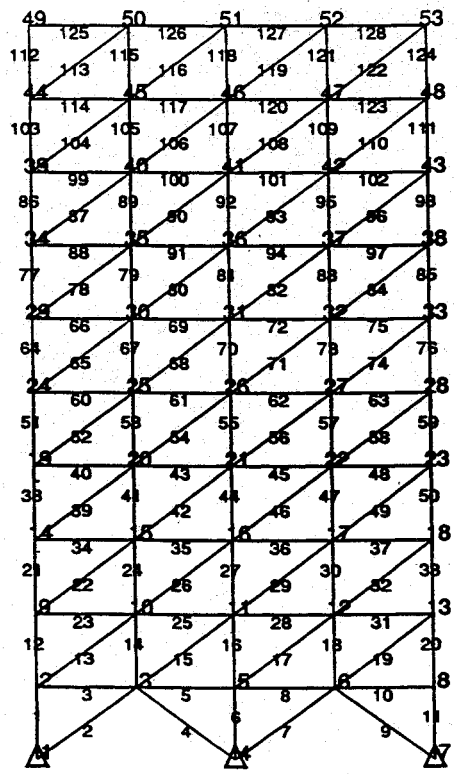

Fig. 6 Example truss structure.

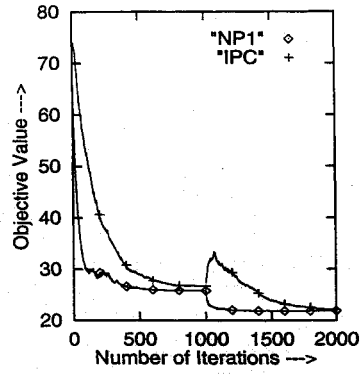

Fig. 7 Convergence plots for optimization using SA with two shakeups run on a Gould NP1 and a sun sparc station IPC.

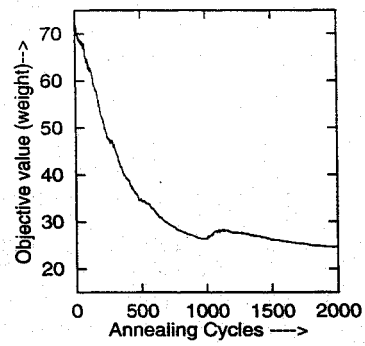

Fig. 8 Convergence of parallel SA on an eight-node configuration of the iPSC/860 parallel computer (with two-stage shakeup).

B. Parallel Solution

For the successful porting of a serial program to a parallel platform, the parallel version of the algorithm should first be executed on a single processor (node). This is for verification purposes and is done by suppressing all internode communications. In this light the 128 bar example problem was run with 3 shakeups, each consisting of 2000 annealing cycles on a single node, and virtually the same results were obtained as with the serial computers. Although this process took longer than the parallelized version, it achieved the convergence effortlessly.

The parallelized algorithm was then run on 8 nodes with 2 shakeups, each consisting of 1000 cycles, and the progression of the annealing process is shown in Fig. 8. Some improvement was observed after the second stage of shakeup, and no significantly better result was expected with more stages. The weight of the structure was minimized from 72.3 to 24.6 lb. Further reduction in weight, although possible, would require an increased computational effort that could not be justified. This slightly suboptimal achievement is due to processor level decisions with only one internode communication at the end of each iteration. Inadvertent constraint violation on random nodes and subsequent inclusion of such results in the assembled one prevented the parallel solution from approaching too close to the highly bounded optimum. This was not critical for the progression of annealing or in locating the vicinity of the solution. Two possible strategies can possibly be used to remedy the situation. One is to move the final design to a single node configuration and run it serially, where a closer optimum can be found, albeit slowly. The other is to continue the optimization of the design using a traditional gradient-based algorithm, assuming that the solution is already in the neighborhood of the global solution. None of these strategies is implemented at this time.

C. Parallel Performance

On a multi-user system, the user gets only a small share of the attention of the CPU. Three different timings for a program execution can be obtained as 1) real time, clock time, includes all activities and idle times; 2) user time, time slice for the users program execution, includes all I/O; and 3) system time, the CPU time. The system time, being the shortest, always looks very attractive. But it excludes all

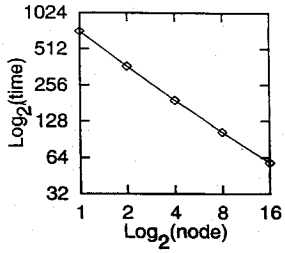

Fig. 9 Scalability plot of the parallel SA algorithm running on the iPSC/860 MIMD parallel computer.

I/O and times taken by the operating system, which is inevitable in any system. On the other hand, iPSC/860 times include CPU times as well as those taken by I/O and the operating system. Therefore, only the user time is valid for comparing with those of iPSC/860. All timings were obtained without the use of fast concurrent file system (CFS) data storage of the iPSC/860. To demonstrate the relative performance of the iPSC/860 parallel computer, the execution time on a 8-node configuration for 1000 iterations was compared with the times taken by the Gould NP1 and Sun workstation. It required 34,100 s on the Gould NP1, 11,180 s on the Sun IPC, and 1090 s on the iPSC/860 (8-node configuration). The computational advantage of the iPSC/860 parallel computer, such as the speed-up achieved, can be seen to be very encouraging.

Scalability of a parallel code measures how well it responds to varying size configurations of the parallel computer. The scalability is also heavily dependent on the uniformity of loading of the nodes by the algorithm as well as the fraction of the code that is parallelized. To test the scalability of the parallel SA algorithm, as implemented on the iPSC/860 system, it was run for 100 cycles using various size configurations of the parallel computer. The results are shown in Fig. 9. Note that only a configuration of 2^n nodes is allowed on a hypercube, where n is the dimension of the current cube. From the plot it is evident that the scalability remained close to being linear, although higher size configurations could alter this significantly.

VI. Conclusion

Simulated annealing is a novel, discrete, and robust optimization method that could be used effectively for structural design optimization. It is also very suitable for implementation on the MIMD parallel processing environment. The performance characteristics of simulated annealing for structural optimization are demonstrated by using both serial and parallel environments. A brief overview of the various parallel architectures and their connectivities are presented. It is found to be conceptually straightforward and efficient to port a serial simulated annealing algorithm to the parallel platform having an MIMD architecture. The parallel implementation is made problem independent so that the user is exempted from any parallel programming task. This is possible with the SA algorithm as subsets of design variables could be processed by various processors concurrently with periodic data exchange. The present parallel implementation required a minimal amount of internode communication. Moreover, a good load balance among the nodes was achieved, as indicated by the scalability plot. Results presented show that even the 16-node iPSC/860 system achieves a handsome speed-up when compared with the dual processor Gould NP1 mainframe sequential computer. For the iterative solution of structural design problems, the MIMD architecture shows a good promise for ease of use and computational efficiency. Because of simulated annealing's data parallel implementation, the iPSC/860 was quite a natural choice as the parallel platform for porting structural design problems. This research indicates that parallel processing, specially on an MIMD system, is a viable and promising option for large-scale structural optimization. A guideline is also provided that shows the expected performance of parallel computers. Other optimization algorithms also could possibly be implemented on MIMD computers for parallel execution with similar computational advantages.

References

[1]Heath, M. T., *Hypercube Multiprocessors*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[2]Davidson, D. B., "A Parallel Processing Tutorial," *IEEE Antennas and Propagation Society Magazine*, 1990, pp. 6–19.

[3]Ragsdale, S., *Parallel Programming*, McGraw–Hill, New York, 1991.

[4]Seigel, H. J., *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA, 1985.

[5]Farhat, C., "Which Parallel Finite Element Algorithm for Which Architecture and Which Problem," *Computational Structural Mechanics and Multidisciplinary Optimization*, edited by R. V. Grandhi, W. J. Stroud, and V. B. Venkayya, American Society of Mechanical Engineering, New York, AD-Vol. 16, 1989, pp. 35-43.

[6]Adeli, H., and Kamal, O., "Optimization of Large Structures on Multiprocessor Machines," *Proceedings of the Second IEEE Workshop on Future Trends of Distributed Computing Systems* (Cairo, Egypt), 1990, pp. 290–295.

[7]El-Sayed, M. E. M., and Hsiung, C.-K., "Optimum Structural Design with Parallel Finite Element Analysis," *Comput. Struct.*, Vol. 40, No. 6, 1991, pp. 1469–1474.

[8]Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, 1983, pp. 671–680.

[9]van Laarhoven, P., and Aarts, E., *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Boston, MA, 1987.

[10]Dixon, L. C. W., and Patel, K. D., "The Place of Parallel Computation in Numerical Optimization IV, Parallel Algorithms for Nonlinear Optimization," Numerical Optimization Center, The Hatfield Polytechnic, TR 125, 1982.

[11]Dixon, L. C. W., Patel, K. D., and Ducksbury, P. G., "Experience Running Optimization Algorithms on Parallel Processing Systems," Numerical Optimization Center, The Hatfield Polytechnic, TR 132, 1983.

[12]Roussel–Ragot, P., and Dreyfus, G., "Problem Independent Parallel Implementation of Simulated Annealing: Models and Experiments," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 8, 1990, pp. 827–835.

[13]van de Geijn, R. A., "Massively Parallel LINPACK Benchmark on the Intel Touchstone Delta iPSC/860 System," Univ. of Texas, Computer Science Rept. TR-91-28, 1991.

[14]Lundy, M., and Mees, A., "Convergence of an Annealing Algorithm," *Mathematical Programming*, Vol. 34, 1986, pp. 111–124.

[15]Hajek, B., "Cooling Schedules for Optimal Annealing," *Mathematics of Operations Research*, Vol. 13, No. 4, 1988, pp. 563–571.

[16]Svensson, B., "Substructuring Approach to Optimum Structural Design," *Computers and Structures*, Vol. 25, No. 2, 1987, pp. 251–258.

[17]Sobieszczanski-Sobieski, J., James, J. B., and Riley, M. F., "Structural Sizing by Generalized, Multilevel Optimization," *AIAA Journal*, Vol. 25, No. 1, 1987, pp. 139–145.

[18]Mueller-Slany, H. H., "Application of Optimization Procedures to Substructure Synthesis," *Proceedings of the International Conference on Engineering Optimization in Design Process* (Karlsruhe, Germany), Lecture Notes in Engineering, No. 33, Springer–Verlag, Berlin, Germany, 1990, pp. 267–274.