IEEE Trans. on Computer-Aided Design, to be published

A PROBLEM-INDEPENDENT PARALLEL IMPLEMENTATION OF SIMULATED ANNEALING : MODELS AND EXPERIMENTS

P. ROUSSEL-RAGOT and G. DREYFUS, SENIOR MEMBER, IEEE

ABSTRACT
We suggest a problem-independent parallel implementation of the simulated annealing algorithm which is guaranteed to exhibit the same convergence behaviour as the serial algorithm. We introduce two modes of parallelization, depending on the value of the temperature, and we derive statistical models which can predict the speedup for any problem, as a function of the acceptance rate and of the number of processors. The performances are evaluated on a simple placement problem with a Transputer-based network, and the models are compared with experiments.

INTRODUCTION

The simulated annealing algorithm [1] is a flexible and powerful optimization method which has proven successful in the search of optimal or near-optimal solutions for complex engineering applications, such as the placement of components in integrated circuits. This nondeterministic approach is particularly well adapted to problems for which conventional methods get trapped in local minima, for it allows transitions of the system that increase the cost function to be minimized. The probability that such a transition will be accepted is controlled by a parameter, called the temperature. Finely tuned cooling schedules [2,3] may ensure fast convergence to near-optimal solutions, but, as the complexity of the problem grows, the best schedule may still require a very large computational time.
In this paper, we propose a problem-independent, parallel simulated annealing algorithm which guarantees the same quality of convergence as the sequential algorithm, and a statistical model of its behaviour is derived. This study provides a straightforward way to implement in parallel any sequential schedule and to evaluate the expected acceleration factor as a function of the number of processors used in parallel.

I. SUMMARY OF PREVIOUS APPROACHES TO PARALLEL SIMULATED ANNEALING.

In an optimization problem, one tries to find the best possible state, or configuration, of a system, according to a given cost function, often referred to as the "energy". The simulated annealing algorithm is a variant of an iterative improvement method: starting with an initial state, the algorithm generates a sequence of attempted perturbations, usually termed "elementary moves".

The moves improving the cost function to be minimized are accepted (as they are in classical methods), and the moves increasing the energy are accepted with a probability which depends on the value of a control parameter, the temperature. The value of the temperature is decreased stepwise; the length of each temperature step is controlled by rules which will be described below.
The implementation of simulated annealing on a multiprocessor is not straightforward, because of the sequential nature of the method. Simulated annealing is commonly described as a sequence of homogeneous Markov chains: each computation step of a chain starts only when the previous step is completed. This is the condition for the whole process to lead to a unique feasible configuration.
In order to reduce the computation time required, two kinds of parallelism can be used:
i- a parallelism in the evaluation of each move: the computation of a given step of the Markov chain depends only on the configuration of the system before the step and is performed without further interaction with the other steps; therefore, each move evaluation can be parallelized inasmuch as the computations of the variation of the cost function and of the acceptance criterion can be made parallel. This kind of parallelism, which is completely problem-dependent, will not be discussed here.
ii- a global parallelism on the Markov chain level, which can obviously be combined with the first kind of parallelism, if necessary.
Several attempts to parallelize the simulated annealing algorithm were reported in the literature. The differences lie in the way in which the problem is implemented in parallel, and the main issues are
i- the convergence conditions of the parallel algorithm,
ii- the dependence of the parallelism on the problem to be solved.
The placement problem - which is used as a test example in the present paper - is of central importance in Computer Aided Design. The approach to the parallel implementation of this problem suggested by Casotto et al. [4] consists in partitioning the set of cells to be placed into a number of subsets equal to the number of processors avalaible; each subset is assigned to a given processor. The processors run asynchronously as long as the moves occur in a given set of cells; exchanges of cells from different sets are allowed, but imply that one of the two processors involved stops. The moves within a set of cells are translations, rotations or exchanges of the cells. This method is well adapted to problems with short-range interactions. The asynchrony of the processors implies that each processor never knows exactly what the current configuration is, which introduces some chaos in the determination of the energy. This method was implemented on a Sequent Balance 8000 with up to 8 processors running in parallel. Casotto et al. [5] also experimented massive parallelization of the simulated annealing algorithm on the Connection Machine.
Clusters of cells are also used by Mallela and Grover [6] in order to reduce the number of cells to be placed in each sub-problem. The placement of the cells in each cluster involves a reduced search space, thus a reduced computation time, each cluster being evaluated in parallel if desired.
Another approach of the cell placement problem is suggested by Darema et al [7]; in this case, each processor evaluates one perturbation of the Markov chain, under the condition that two processors

are not allowed to move the same cells simultaneously; therefore, there is no conflict between processors and the final configuration is always a valid one. Whenever a perturbation is accepted, the configuration of the cells is updated, regardless of the moves which are being computed. At low temperature, when the acceptance ratio is low, this does not introduce a large bias as compared to the sequential method. At higher temperature, the behaviour of the parallel method deviates significantly from that of the sequential method. The measurements in this study were performed in a parallel emulation environment, allowing simulation of a shared memory multiprocessor system with up to 64 processors.

A comparable approach was suggested by Brouwer and Banerjee [8] for channel routing: sets of adjacent tracks are assigned to parallel processors and nets are moved between tracks which are assigned to a given processor; information about the moves are then broadcast to other processors to update their data structures.

A different approach was used by Kravitz et al [9,10]; they introduce the notion of serializable subset of moves. A set of moves is serializable if the moves do not interact with each other. If the moves of such a subset are evaluated in parallel, the result is the same as it is in sequential. But the determination of these subsets is more and more difficult and time consuming as the number of processors increases. The suggested solution to this problem is the following: they consider the "simplest serializable subset", where only one of the accepted moves is accepted, while all the rejected moves are counted. Then, the parallel algorithm is quite different from the sequential one at high temperature since the acceptance ratio is not the same. This issue will be discussed in detail in the next section. The experiments were performed on a DEC VAX 11/784, consisting of four VAX 11/780 processors connected to a shared 8Mbyte memory.

Yet another approach was suggested by Aarts et al [11] who used, at high temperature, different processors to work on different short Markov chains. When the temperature is changed, one of the final configurations is used as the initial configuration of the next temperature step. As the temperature decreases, less and less Markov subchains are evaluated in parallel, while more and more processors are used for the generation of each subchain. At low temperature, all processors evaluate a single Markov chain. The behaviour of this method at high temperature is very different from the behaviour of the sequential algorithm since the configurations transmitted from one Markov chain to another are obtained after a shorter number of steps than in the sequential chain. Aarts used a parallel machine consisting of fifteen Motorola 68000 microprocessors with local memory and a shared 8 Mbyte memory.

II. PRINCIPLE AND MODELS.

The present section describes the principle of the parallel implementation of simulated annealing that we suggest. We first explain and justify the principle of the parallelization. We subsequently propose a statistical model of this stochastic parallel computation.

1) General considerations

We first define the acceptance rate $\chi(T)$ as the ratio of the number of accepted moves to the number of attempted moves, averaged over a given temperature One of the salient features of simulated annealing is the decrease of $\chi(T)$ as the temperature is reduced. This is due to two facts: first, the system approaches a minimum and it is unlikely that a move decreases the energy; moreover, the value of $\exp(-\Delta E/T)$, where ΔE is the variation of the cost function and T is the temperature, becomes very small, so that the probability of accepting a move which increases the energy vanishes. Most parallel simulated annealing methods take advantage of this fact: in the low temperature regime, when the acceptance rate χ is small, one can use a number K of processor such that $K<1/\chi$. Thus, at most one move will be accepted while K moves are evaluated, so that the computation time in the parallel mode can be expected to be smaller than the computation time in the sequential mode by a factor of the order of K.

However, the computation time is not the only performance criterion. One wants, of course, to obtain a valid solution, and hopefully the optimal one (or one of the optimal ones). Therefore, the convergence of the algorithm is of central importance. Several theoretical studies of the sequential algorithm have been published [12,13], but the theory is much less developed for parallel simulated annealing. However, some parallelization schemes may hamper greatly the convergence of the algorithm, and even make it impossible in some cases. For instance, in the context of massively parallel implementations of simulated annealing for image processing, it was proved by A. Trouvé (private communication), that the algorithm may converge towards states of high energy of the system.Therefore, it seems desirable to design a parallel scheme which complies with the convergence conditions of the sequential algorithm, so that one can capitalize on the accumulated knowledge related to sequential simulated algorithm.

In the following, we suggest a parallel simulated annealing scheme which

i- is problem-independent

ii- has the same convergence properties as the sequential algorithm.

2) Principle

As stated above, we use K processors in parallel, each of them evaluating one move. Each processor has its own memory. We want to design a parallel scheme which is equivalent to the sequential one, as far as convergence is concerned.

The usual, sequential, Metropolis algorithm at fixed temperature T consists in generating a Markov chain of states, the energies of which would have a Boltzmann distribution if the chain were infinitely long; since the chain is of finite length L, the resulting distribution will actually be "close to" the Boltzmann distribution. We want to find a parallel scheme which generates a Markov chain, the states of which have the same probability distribution as the sequential chain, all other parameters being equal. This guarantees that the convergence behaviour of the parallel algorithm will be similar to the behaviour of the sequential one. We shall see in the next section that the acceptance rate $\chi(T)$ is of central importance. Its value leads us to consider two different regimes:

- a low temperature regime: if $\chi(T)<1/K$, less than one move out of K will be accepted; thus, the scheme proceeds as follows: the processors attempt moves on their own, asynchronously, in parallel, until one of the K processors accepts a move; when an accepted move is found, the processors are synchronised, their memories are updated with the new configuration and the next evaluation step takes place.
- a high temperature regime ($\chi(T)>1/K$): in this regime, each processor is allowed to evaluate one move only and waits until all the other processors complete their evaluation. Then, one of the accepted moves is chosen at random, the processor memories are updated with the new configuration and the next evaluation step takes place. The reason why we choose one of the accepted move at random, instead of choosing the first accepted move, is the following: since the computation time of a single move can vary substantially, choosing the first move would greatly favour the short computations (for instance, moves of weakly connected blocks, or downhill moves which do not require the computation of the exponential).

In addition to the above mentioned K "slave" processors, the scheme requires one "master" processor which monitors the annealing schedule, chooses the accepted move in the high temperature regime, updates the memory of each processor and keeps track of statistics.

3) Models

In all the following, we use the standard annealing schedule, whereby the temperature is decreased stepwise according to $T_{n+1} = \alpha T_n$, where T_n is the nth temperature and α may range from 0.9 to 0.99. We denote by

L_a, the maximum number of accepted moves at a given temperature,

L_t, the maximum number of attempts at a given temperature,

τ_o, the average computation time necessary to evaluate one move in the sequential mode.

The meaning of τ_o is clear if only one type of elementary move is used; otherwise, it would represent the average value of the computation time of the various types of moves occuring during annealing. In both cases, it can be estimated as the duration of a temperature step divided by the number of attempted moves. If the perturbations vary with the temperature (for example, if exchanges of blocks occur mainly at high temperatures and translations mainly at low temperatures), the value of τ_o will vary with temperature.

The temperature is decreased either when the number of accepted moves at the current temperature reaches L_a, or when the number of attempted moves at the current temperature reaches L_t, whichever limit is reached first.

Note that τ_o, the average computation time to evaluate one move in the sequential mode, is not exactly the same as that in parallel, because the master processor can take care of the necessary statistics while the slave processors evaluate the moves. Therefore, τ_o is an upper limit of the average computation time for one move in the parallel implementation.

a) Model of the high temperature mode

In the high temperature mode, each processor evaluates one move, and all processors are synchronized at the end of each evaluation; we denote by τ_r the average overhead due to communications with the K slaves and their synchronization: therefore, the average time necessary for the K processors to perform one evaluation is $\tau_o + \tau_r$. Note that τ_r takes into account the fact that the moves may be of different durations, so that the time to complete a parallel evaluation is equal to the duration of the longest attempted move.

Since the length of the Markov chain depends on the number of accepted moves and/or on the number of attempted moves, we first have to evaluate these quantities. Assume that, after one parallel evaluation of K moves, r moves out of K are rejected; K-r moves are found acceptable, but only one of them will actually be accepted in the Markov chain, the other ones being discarded. The ratio of the number of accepted moves to the number of attempted moves, in the serial mode, would be (K-r)/K; however, in the parallel mode, only one move is accepted. How do we construct a chain which has the same acceptance ratio as the sequential one? The procedure is as follows: we number the processors in an arbitrary, but definite order, from 1 to K. We denote by n, the number of the first processor in the list which accepts a move and we construct the Markov chain with the n first moves in the list. A Markov chain of n-1 rejected moves, followed by one accepted move, would have been found with the same probability in the sequential mode. It can be shown that the average value of n is equal to $n^*=(K+1)/(K-r+1)$ (see appendix).

If all K processors reject their attempted moves, the number of the first processor cannot be found using this approach, but since there is no accepted move, the number of attempted moves to be taken into account is clearly K.

To summarize, when K evaluations are performed in parallel:

- if at least one move has been accepted, one of the accepted moves is chosen randomly and we consider that $n^*=(K+1)/(K-r+1)$ moves have been attempted,
- if no move has been accepted, we consider that K moves have been attempted.

This provides a good estimate of the effective number of attempted moves since, on the average, the ratio of the number of accepted moves to the effective number of attempted moves in the parallel mode is equal to the sequential acceptance rate. This can be shown as follows.

The average number n^*_a of accepted moves for one parallel evaluation is equal to the ratio of the number of parallel evaluations for which at least one move is accepted (since, in this case, one move is taken into account) to the total number of parallel evaluations. The probability that i moves

out of K are accepted is equal to
$\binom{K}{i} . (1-\chi)^{K-i} . \chi^i$;
the probability that all K moves are rejected is $(1-\chi)^K$. Thus,

$$n_a^* = \sum_{i=1}^{K} \binom{K}{i} \chi^i (1-\chi)^{K-i} = 1-(1-\chi)^K .$$

When i moves out of K are accepted, the number of attempted moves is taken equal to K+1/(i+1), so that the effective number of attempted moves n^*_t in the parallel mode is equal to

$$n_t^* = (1-\chi)^K . K + \sum_{i=1}^{K} \binom{K}{i} \chi^i (1-\chi)^{K-i} \frac{K+1}{i+1} .$$

Therefore, the acceptance rate in the parallel mode is equal to

$$\frac{\sum_{i=1}^{K} \binom{K}{i} \chi^i (1-\chi)^{K-i}}{(1-\chi)^K . K + \sum_{i=1}^{K} \binom{K}{i} \chi^i (1-\chi)^{K-i} \frac{K+1}{i+1}} .$$

This ratio can be shown after some algebra to be equal to χ. Thus, we obtain the same convergence behaviour in the parallel mode and in the sequential mode since the parallel algorithm has the same transition probability matrix as the sequential one.
We now evaluate the total effective number of attempted moves N^*_t, and the total number of accepted moves, actually taken into account, N^*_a, once N parallel evaluations of K moves have been performed.
The average effective number of attempts is given by

$$N_t^* = N . \left[(1-\chi)^K . K + \sum_{i=1}^{K} \binom{K}{i} \chi^i (1-\chi)^{K-i} \frac{K+1}{i+1} \right]$$

or equivalently

$$N_t^* = N . \frac{1-(1-\chi)^K}{\chi} .$$

The limit of L_t attempted moves is reached after a number N_t of parallel evaluations of K moves which is given by

$$N_t = L_t . \frac{\chi}{1-(1-\chi)^K} .$$

The number N^*_a of accepted moves actually taken into account is equal to the number of parallel evaluations of K moves leading to at least one acceptable move, hence
$N^*_a = N[1-(1-\chi)^K]$.
Therefore, the limit of L_a accepted moves is reached after a number N_a of parallel evaluations of K moves which is given by

$$N_a = \frac{L_a}{1-(1-\chi)^K} .$$

Therefore, in the high temperature mode, the number of parallel evaluations at a given temperature is
$N_p = \min (N_t, N_a)$.
The corresponding computation time is
$t_p = N_p (\tau_o + \tau_r)$.

In the serial mode, the computation time is

$t_s = \tau_o . L_a / \chi$	if the L_a limit is reached first,
$t_s = \tau_o . L_t$	if the L_t limit is reached first.

Therefore, whichever limit L_a or L_t is reached first,

$$\frac{t_p}{t_s} = \frac{\chi}{1-(1-\chi)^K} \left(1 + \frac{\tau_r}{\tau_0} \right) \qquad (1) .$$

Note that lim $(t_p / t_s) = 1 + \tau_r/\tau_o$ when $\chi \text{--->} 1$ and lim $(t_p / t_s) = 1/K.(1+\tau_r/\tau_o)$ when $\chi \text{--->} 0$.

At high temperature, the efficiency is low because the parallel mode skips rejected moves and only few moves are rejected at high temperatures when the acceptance rate is high; at low temperature, the computation time is roughly divided by the number of processors, as expected, if the overhead time τ_r is small compared to τ_o.
The average value of τ_o is known from the serial implementation of the simulated annealing algorithm. The determination of τ_r is not straightforward and depends on the problem. If τ_o is constant, τ_r may be approximated conservatively by K times the communication time: if the K parallel computations end at the same time, K successive communications will be required for the master to know all the results and restart the slaves.

b) Model of the low temperature mode

In the low temperature mode, each processor evaluates moves independently until one processor out of K accepts a move. At the end of each individual evaluation, the processors send asynchronously to the master the result of their attempt; if no move was accepted by any processor since the previous communication, another move is attempted. If one move was accepted, the memories of the slave processors are updated and the processors are synchronized. In this mode, all the rejected moves are counted as steps towards equilibrium.

In order to model the behaviour of this low temperature mode, it is necessary to evaluate the number of moves required for one move to be accepted. This can be done in two ways: Aarts et al [11] evaluate the number of parallel calculations required. Their estimation leads to a number of parallel calculations equal, on the average, to $1 / 1-(1-\chi)^K$. Since it is easier to estimate the time characteristics of individual moves from the sequential results, we find it preferable to evaluate the number of such moves. One configuration is accepted on the average when $1/\chi$ moves are evaluated. The process has then performed $1/\chi$ steps towards equilibrium in the serial mode. Since we want to obtain a feasible configuration of the system, if another move isaccepted by one of the K-1 other processors, we do not take it into account: the perturbations are chosen at random and, if, for example, one block is exchanged in two different moves, the resulting configuration is not a valid one since this block would be placed in two different locations. When the processors are synchronized, $1/\chi$ + K-1 moves have been evaluated and, on the average $(1/\chi + K-1).\chi$ have been accepted. Since we discard all the accepted moves, but one, we count only

$(1/\chi+K-1) - (1/\chi+K-1).\chi + 1 = 1/\chi + (K-1).(1-\chi)$

steps in the Markov chain .

If τ_m is the time required to obtain one accepted move in parallel, we can model the behaviour of the low temperature mode as follows:

i- when L_a is reached first,

$t_p = L_a . \tau_m$ and $t_s = L_a/\chi . \tau_o$, thus

$$\frac{t_p}{t_s} = \chi \frac{\tau_m}{\tau_o} \qquad (2) .$$

ii- when L_t is reached first,

$$t_p = \frac{L_t}{1/\chi + (K-1)(1-\chi)} \tau_m \qquad (3) .$$

and

$t_s = L_t.\tau_o$, thus

$$\frac{t_p}{t_s} = \frac{1}{1/\chi + (K-1)(1-\chi)} \frac{\tau_m}{\tau_o} .$$

Note that for K=1, one has $\tau_m = \tau_o/\chi$, so that $t_p = t_s$, as expected.

Here again, the average value of τ_o is known, but the determination of τ_m is not an easy task and depends on the problem. If τ_o is constant and if $1/\chi$ is much larger than K, the value of τ_m can be

approximated by $(\tau_o + \tau_c) / K\chi$, τ_c being the time required by one slave to communicate its result to the master.

III. RESULTS

1) A simple placement problem

We tested our parallel methods and models on a simple placement problem. It consists of a two dimensional array of b^2 chips arranged on a square grid. In the ground state configuration of the system, each chip is connected to its nearest neighbours by two-terminal connections. The elementary move is the exchange of two chips, chosen at random; the cost function is equal to the total length of the wires. The parameters of the standard sequential annealing schedule are the following:

- the initial configuration is chosen at random,
- the initial temperature is chosen so that the acceptance rate is larger than 0.9,
- the temperature is modified when $5.b^2.(b^2 -1)$ moves have been evaluated or when $b^2.(b^2 -1)/2$ moves have been accepted,
- the cooling parameter α is equal to 0.9,
- the simulated annealing process is stopped when the temperature reaches the value 0.2 or when no move is accepted at a given temperature.

This annealing schedule was not intended to be optimal: we only wanted to evaluate the performances of the parallel algorithm as compared to those of the serial algorithm, subject to the same conditions.

2) The Transputers

We implemented the parallel algorithm on a network of Transputers. A Transputer is a 32-bit processor with its own memory. Communications occur only through four high-speed serial links, so that each Transputer can communicate with four neighbour Transputers. One link is used for communications between two Transputers only, so that parallel architectures are implemented simply by connecting the serial links between the pairs of communicating Transputers. T800 Transputers have an on-chip floating-point processor, which T414 Transputers have not. Instructions for communications are available in the software (OCCAM) and no data can be lost since these communications are synchronized. We use one Transputer as the master processor and we perform the computations on Transputers linked to the master. The architecture can be improved so that no Transputer has to be used only for communications, but this does not restrict the validity of our model.

Experiments were performed with 25, 49 and 81 chips on 3 and 6 Transputers. We shall present the most relevant results here, obtained on 81 chips in two cases: the communication time τ_r is small as compared to the computation time τ_o ($\tau_o \cong 4.2$ msec and $\tau_r \cong 0.5$ msec), and the communication and computation times are of the same order of magnitude ($\tau_o \cong 0.6$ msec and $\tau_r \cong 0.3$ msec).

3) Numerical results

Both temperature modes were investigated independently on a complete annealing, although they are not intended to be actually used on the whole temperature range.

We compared the behaviour of the high and low temperature modes to the behaviour of the sequential mode, on 100 different initial configurations. Figure 1a is a plot of the average final energy of each temperature step as a function of temperature; it can be seen that the high temperature mode behaves similarly to the sequential mode, whereas the low temperature mode decreases the energy quickly for high temperature values. This is due to the fact that, in the low temperature mode, the first accepted move is taken into account for updating the system. Since we used T414 Transputers without floating-point computations, the computation of the exponential is very long as compared to the execution of a simple instruction; thus, the first accepted move often happens to be a move which decreases the energy. In addition, the low-temperature mode is not expected to have the same acceptance rate as the sequential algorithm at high temperature: since all rejected moves are counted as steps of the Markov chain, and since the total number of moves at a given temperature is the same as in the sequential mode, the number of accepted moves is lower than in the sequential computation; therefore, the probability of escaping from a local minimum is lower than it should be at the considered temperature, thus the average energy is lower. This introduces a bias which does not affect the final result in this problem, but might do in others; therefore, it is definitely not desirable, in general, to use the low-temperature mode at high temperature. At low temperature, the annealing curve is the same for the three modes. This allows us to switch from high to low temperature mode when the low temperature mode becomes more efficient, still complying with the quality of convergence of the sequential algorithm. The final and initial energy distributions are shown on Figure 1b: no significant difference between the three modes can be observed because of the nature of the problem we investigated; a difference should appear when using finely tuned annealing schedules, since the low temperature mode exhibits significant deviation at high temperature.

The computation times, averaged over 10 experiments, for each temperature step of the annealing process for the sequential and the parallel algorithms, are shown on Figure 2. On all diagrams, the upper three curves are the average duration of a temperature step measured if the temperature is decreased when L_t moves have been attempted; the lower three curves are the average duration of a

temperature step if the temperature is decreased when L_a moves have been accepted. As expected, the duration of a temperature step in the sequential mode is virtually constant in the first case and increases sharply at low temperature in the second case. We find that the acceleration is poor at high temperature: the time required for the parallel algorithms is even higher than the time required for the sequential algorithm when the communication and synchronization time is close to the computation time (Figures 2b and 2c). This results from the high value of the acceptance rate: few moves are rejected, so that the parallel mode is not efficient. In contrast, at low temperature, the duration is almost divided by the number of processors. It can be seen on Figure 2c that the average duration of a temperature step, when the limit L_t is used, is small at high temperature for the low temperature mode; this is due to the fact that, in this mode, as mentioned before, the energy decreases quickly at high temperature; thus, the acceptance rate decreases too, and the number of rejected moves is high. Moreover, in this mode, all the rejected moves are counted as steps of the Markov chain. When the L_a limit is used, the overall annealing time is divided by 2 with 3 processors when the communication time is large, and by 2.5 when it is small. This value depends strongly on the annealing schedule: if more time is spent at low temperature, as is frequently the case in real optimization problems, the overall speedup factor is higher.

Since we use a linear scale for temperature, whereas the decrease of the temperature is geometric, the overall gain in annealing time does not appear clearly; thus we present, on Figure 3, the cumulated annealing time for the sequential algorithm and for the parallel algorithm, using the L_a limit in the case where τ_r is close to τ_o. The total time is divided by 2 with this parallel method when 3 Transputers are used and by 3 when 6 Transputers are used.

Figure 4 exhibits a very good agreement between the measurements performed in the parallel temperature mode and the estimated values of t_p/t_s when the L_a limit is reached first (relations 1 and 2), and when the L_t limit is reached first (relations 1 and 3). The acceleration is higher when the L_t limit is used in the low temperature mode, but, since each temperature step is much longer than in the case of the L_a limit, it is definitively worthwhile using the L_a limit when it is reached first. To compute the estimates from the model, we evaluated $\tau_o + \tau_r$ as the average duration of a temperature step divided by the number of parallel evaluations, and τ_m as the average duration of a temperature step divided by the number of accepted moves. The average value of $\chi(T)$ was estimated from the sequential results.

In the low temperature mode, communications were implemented between the working processors and the master processor each time a move is attempted. This was required for measuring the number of moves actually attempted in order to evaluate the model. For real implementation of this parallel algorithm, only synchronization on the accepted moves is required, so that the accelerations obtained in our example are the worst case. A further modification of the parallel mode can be used:

the low temperature mode is not used at high temperature because the first accepted move is the shortest to compute, which may introduce a bias, as observed in our experiments. If the processors are not synchronized when a move is accepted, that is, if the processors communicate the result of the move they have evaluated and start a new move as soon as their configuration is actualized, then, the computations become asynchronous, and the first accepted move may not be the one which is the shortest to compute, since the processors start their individual evaluations at different times. Thus, the bias observed in our experiments should vanish. This asynchronous mode could then be used for the whole temperature range and this method should further reduce the computation time, because no synchronization time is required. Nevertheless, only the rejected moves evaluated before one move is accepted should be counted in the Markov chain in order to preserve the quality of convergence of annealing, since the acceptance rate must be preserved.

CONCLUSION

We propose a problem-independent parallel implementation of the simulated annealing algorithm, which guarantees the same quality of convergence as the sequential algorithm. The parallel algorithm consists of two modes: one is intended to be used at high temperature (at least one move is accepted when K moves are evaluated) and the other one is to be used at low temperature (at most one move is accepted). Statistical models of both modes are derived and compared to experiments on a simple placement problem, implemented on a network of Transputers; the architecture of the system consists of a "master" processor linked to K "slaves" processors. These models are expressed as functions of the acceptance rate, which enables an estimation of the acceleration for any optimization problem. They take into account the fact that, depending on the implementation of the sequential algorithm, the length of the Markov chain for each temperature step may be taken either equal to a given number of accepted moves or equal to a given number of attempted moves. The condition for switching from the high temperature mode to the low temperature mode will depend on the number of processors used, and on the length of the Markov chains.

As mentioned above, it is possible to increase the speedup if the synchronization between the slaves is used only when a move is accepted. Thus, the speedup evaluated from the models is the worst case and can be improved for real implementations.

Acknowledgments: The authors are very grateful to A. Trouvé and O. Catoni for their critical comments on the derivation of the effective number of moves. They wish to thank the Groupe de Soutien aux Utilisateurs of the CIRCE Computer Center.

Appendix.

We denote by K the number of processors numbered from 1 to K, and by i, the number of accepted moves among the K moves attempted in parallel. We show in the following that the average value of the number of the first processor in the list which accepts a move is equal to $n^*=(K+1)/(K-r+1)$ [14].

We denote by σ the number of the first processor in the list which accepts a move.

The probability that σ is equal or higher than n is

$$P(\sigma\geq n)=\frac{\binom{K-n+1}{i}}{\binom{K}{i}}=\frac{(K-n+1)(K-n)\ ...(K-n-i+2)}{K(K-1)\ ...(K-i+1)}$$

Thus,

$$P(\sigma=n)=P(\sigma\geq n)-P(\sigma\geq(n+1))=\frac{\binom{K-n+1}{i}-\binom{K-n}{i}}{\binom{K}{i}}$$

which can be written as follows

$$P(\sigma=n)=\frac{i(K-n)(K-n-1)\ ...(K-n-i+2)}{K(K-1)\ ...(K-i+1)}$$

The expectation value of σ, n^*, is equal to

$$n^*=\sum_{n=1}^{K-i+1} n.P(\sigma=n)=\sum_{n=1}^{K-i+1} P(\sigma\geq n)$$

Thus,

$$n^*=\sum_{n=1}^{K-i+1}\frac{(K-n+1)(K-n)\ ...(K-n-i+2)}{K(K-1)\ ...(K-i+1)}=\frac{1}{K(K-1)\ ...(K-i+1)}\sum_{m=i}^{K} m(m-1)\ ...(m-i+1)$$

The summation

$$\sum_{m=i}^{K} m(m-1)\ ...(m-i+1)$$

is equal to the ith derivative for x=1 of

$$\sum_{n=0}^{K} x^n=\frac{1-x^{K+1}}{1-x}=\sum_{j=0}^{K}\binom{K+1}{j+1}(x-1)^j$$

Thus, the expectation value is

$$n^*=\frac{i!\binom{K+1}{i+1}}{K(K-1)\ ...(K-i+1)}=\frac{K+1}{i+1}$$

If we denote by r, the number of rejected moves, we have i=K-r, and

$$n^*=\frac{K+1}{K-r+1}\ .$$

[1] S. Kirkpatrick, C. Gelatt,Jr. and M. Vecchi, "Optimization by Simulated Annealing", Science, Vol 220, 671-680, 1983.

[2] J. Lam and J-M. Delosme, "Performance of a new annealing schedule", Proceedings of the 25th IEEE Design Automation Conference, 306-311, 1988.

[3] R. Otten and L. van Ginneken, "Stop criteria in simulated annealing", Proceedings of the IEEE International Conference on Computer Design, 549-552, 1988.

[4] A. Casotto, F. Romeo and A. Sangiovanni-Vincentelli, " A parallel simulated annealing algorithm for the placement of macro-cells", IEEE Transactions on CAD, Vol CAD-6, NO 5, 838- 847, 1987.

[5] A. Casotto and A. Sangiovanni-Vincentelli, "Placement of standard cells using simulated annealing on the connection machine", Proceedings of the IEEE International Conference on Computer Design, 350-353, 1987.

[6] S. Mallela and L. Grover, "Clustering based simulated annealing for standard cell placement", Proceedings of the 25th IEEE Design Automation Conference, 312-317, 1988.

[7] F. Darema, S. Kirkpatrick and V. A. Norton, "Parallel algorithms for chip placement by simulated annealing", IBM J. Res. Develop., Vol. 31, No 3, 391-402, 1987.

[8] R. Brouwer and P. Banerjee, "A parallel simulated annealing algorithm for channel routing on a hypercube multiprocessor", Proceedings of the IEEE International Conference on Computer Design, 4-7, 1988.

[9] S. Kravitz and R. Rutenbar, "Multiprocessor-based placement by simulated annealing", Proceedings of the 23th IEEE Design Automation Conference, 1986.

[10] R. Rutenbar and S. Kravitz, "Layout by annealing in a parallel environment", Proceedings of the IEEE International Conference on Computer Design, 434-437, 1986.

[11] R. Jayaraman and R. Rutenbar, "Floorplanning by annealing on a hypercube multiprocessor", Proceedings of the IEEE International Conference on Computer Aided Design, 346-349, 1987.

[11] E. Aarts, F. de Bont, E. Habers and P. van Laarhoven, "'Parallel implementations of the statistical cooling algorithm", Integration, the VLSI journal, Vol. 4, 209-238, 1986.

[12] E. Aarts and P. van Laarhoven, "Statistical cooling: a general approach to combinatorial optimization problems", Philips Journal of Research, Vol. 40, No 4, 193-226, 1985.

[13] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", IEEE Proceedings on Pattern analysis and machine intelligence, 722-741, 1984.

[14] O. Catoni and A. Trouvé, "A propos du recuit simulé synchrone", unpublished.

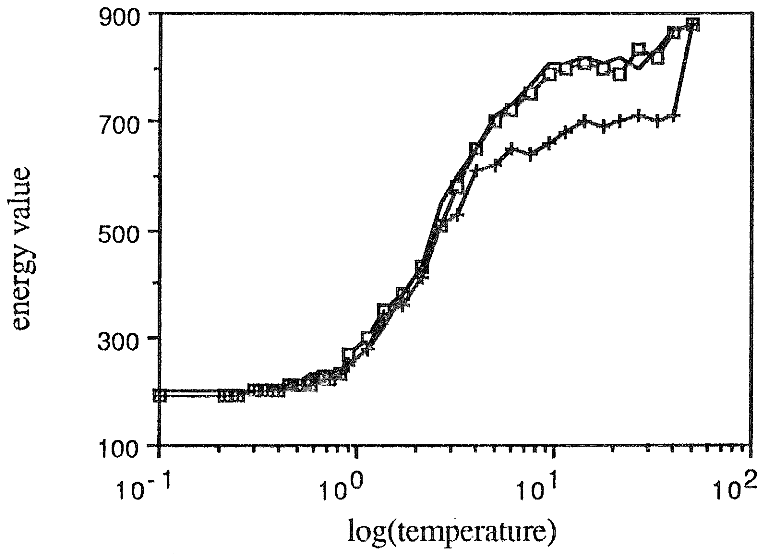

Figure 1a .
Final value of the energy for a temperature step
for the sequential mode (squares)
and the high temperature (dots)
and low temperature (crosses) modes.

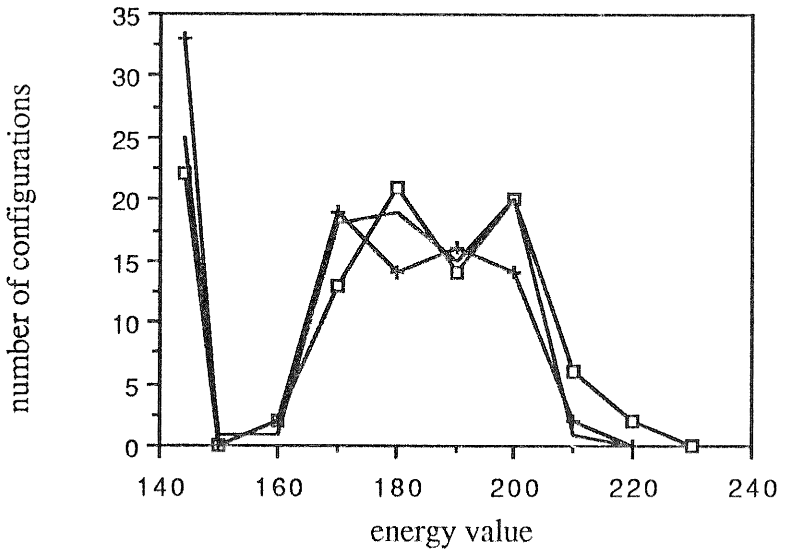

Figure 1b .
Energy distribution of the initial configurations (right-hand peak),
and energy distributions of the configurations
after annealing (left-hand peaks).

Figures 2. Average duration of a temperature step versus temperature;
results obtained in the sequential mode (squares),
high temperature mode (dots)and low-temperature mode (crosses);
upper three curves : temperature decreased when Lt moves have been attempted;
lower three curves : temperature decreased when La moves have been accepted.

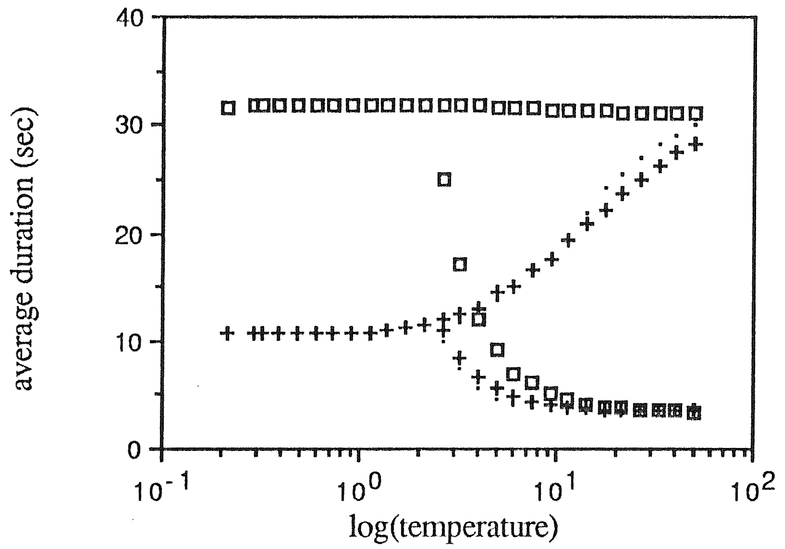

Figure 2a.
Overhead time much smaller than computation time;
measurements performed with three T414 Transputers.

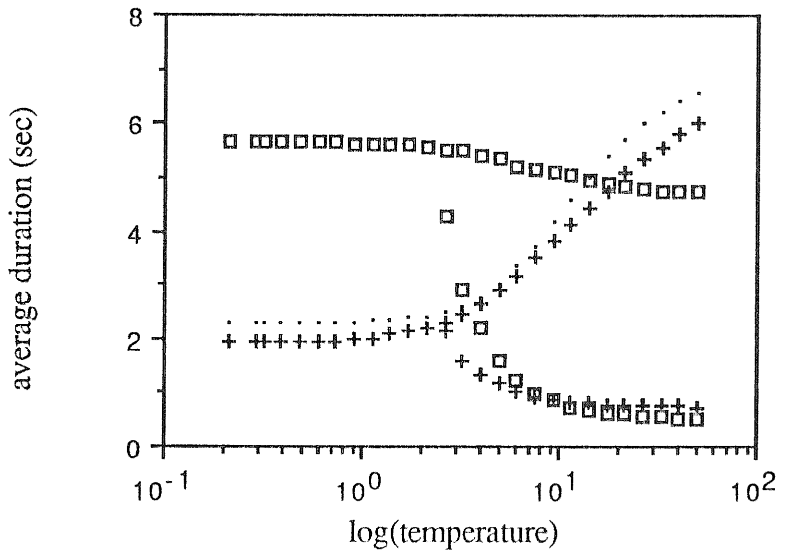

Figure 2b.
Overhead time of the same order of magnitude
as computation time;
measurements performed with three T414 Transputers.

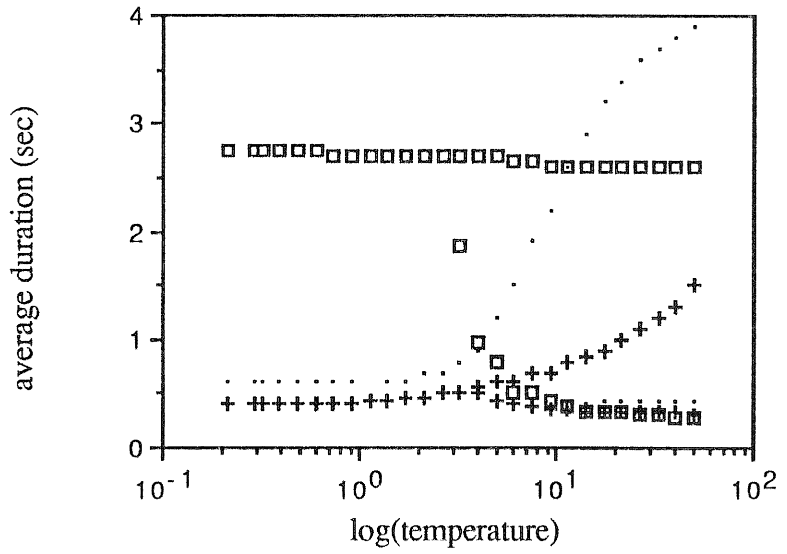

Figure 2c.
Overhead time of the same order of magnitude
as computation time;
measurements performed with six T800 Transputers.

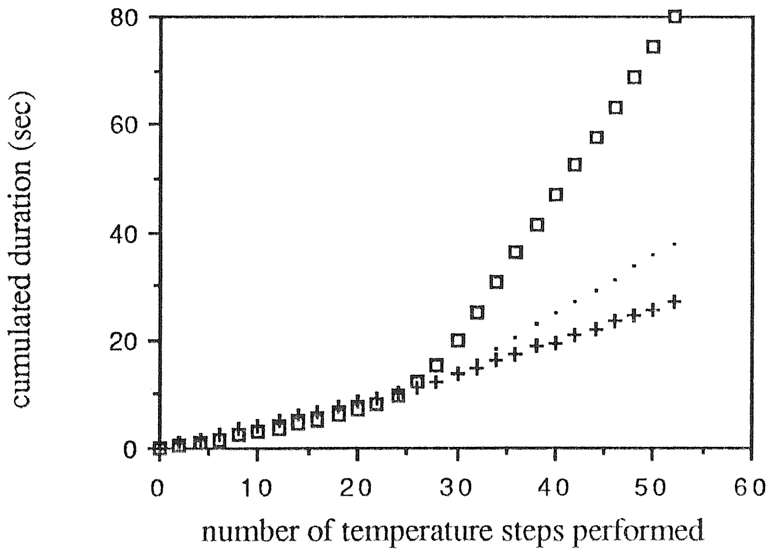

Figure 3.
Cumulated duration of a complete annealing
in the sequential mode (squares)
and in the parallel mode with three (dots) and six (crosses) Transputers;
temperature decreased when La moves have been accepted.

Figures 4. Ratio of the duration of a temperature step in the parallel mode
to the duration of a temperature step in the sequential mode;
measurements performed with three Transputers;
overhead of the same order of magnitude as computation time;
squares: measurements results; +: results computed from the model when the La limit is used;
black squares: measurements results; x: results computed from the model when the Lt limit is used.

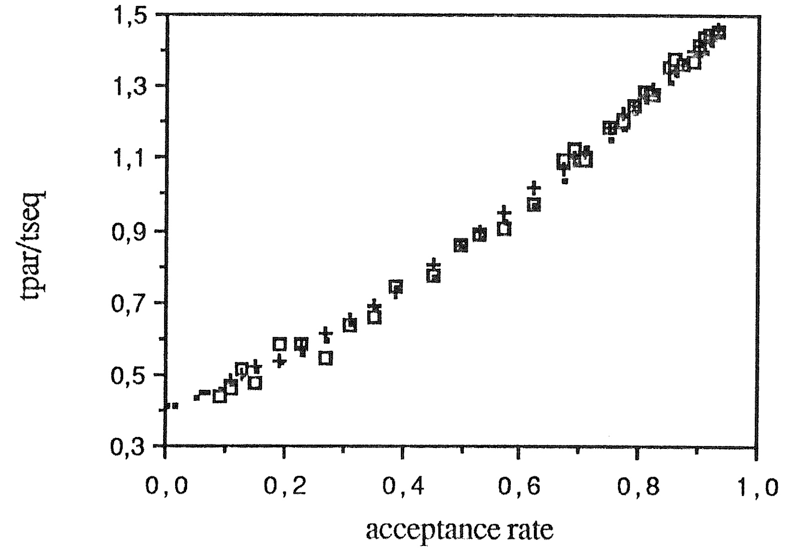

Figure 4a. High-temperature mode.

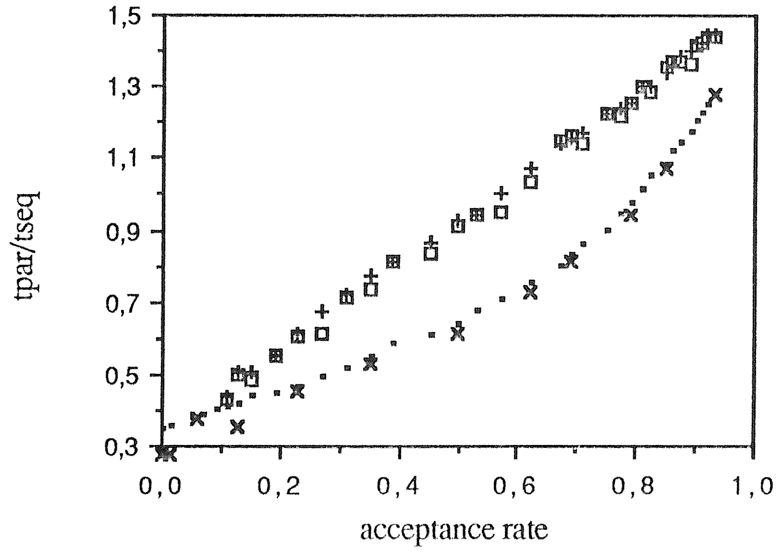

Figure 4b. Low-temperature mode.