Simulated Tempering and Mean Field Annealing for Mapping to Multicomputers

By

Gabriel I. Aghazarian

June 1996

To a rare instructor

To a rare supervisor

To a rare humanist

To Dr. N. Mansour

Yours Truly

Gabriel Aghazarian

Simulated Tempering and Mean Field Annealing for Mapping to Multicomputers

By
Gabriel I. Aghazarian

THESIS

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science
at the Lebanese American University
June 1996

Signatures Redacted

Dr. Nashat Mansour (Advisor)
Assistant Professor of Computer Science
Lebanese American University

Signatures Redacted

Dr. George Nasr
Assistant Professor of Electrical and Computer Engineering
Lebanese American University

Signatures Redacted

Dr. Haidar Harmanani
Assistant Professor of Computer Science
Lebanese American University

Abstract

We worked on two physical optimization algorithms, Mean Field Annealing and Simulated Tempering, to solve the data mapping problem on multicomputers. We studied the effects of user defined parameters on MFA's behavior. These experiments resulted in recommending values to these parameters. Next, we implemented, for the first time, Simulated Tempering to solve the mapping problem. Then, we applied Mean Field Annealing, Simulated Tempering together with Simulated Annealing, Genetic Algorithm and Neural Network on randomly generated graphs. It was found that Neural Network (NN) proved to be the fastest algorithm although the quality of its solution is inferior to that of the Genetic Algorithm (GA), which was superior to all algorithms. Simulated Annealing had the second rank for both execution time and solution quality.

Acknowledgments

I would like to thank all those who have assisted me in accomplishing this work. Special thanks go to Professor Nashat Mansour for his patience, perseverance, continuous encouragement, help and support he provided me. I would like to thank the second readers, Professor George Nasr and Professor Haidar Harmanani. Also I would like to thank the Natural Science Division Chairperson and Committee for the financial assistance.

Special thanks go to my colleagues Jalal Kawash and Khaled Fakih for the valuable comments and discussions, the A.C.C. supervisor Mr. Tarek Dana, and the assistant A.C.C. supervisor Mr. Ali Aleywan for their help.

Table of Contents

Chapter 1

Introduction

Consider an algorithm intended to solve a problem with an underlying data set on a multicomputer. Then the data mapping problem is the problem of partitioning the data into mutually exclusive subsets, each of which is mapped to a processor node in the multicomputer. The aim is minimizing the execution time of the parallel algorithm on the multicomputer.

The assumptions made in this work are the following:

- the multicomputer is a distributed-memory message-passing machine whose processors are connected by a static point-to-point interconnection network.
- the computation model used is loosely synchronous computation model in which processors perform compute-communicate cycles in a SPMD (Single Program, Multiple Data) scheme [Fox et al. 1988].

Under these assumptions, the execution time on the multicomputer is determined by the execution time of the slowest processor. Thus, the data mapping problem is obviously an

optimization problem, and achieving data-parallelism involves nearly-equal distribution of computation workloads over the processors of the multicomputer, as well as the minimization of the amount of their inter-processor communication. The data mapping problem is NP-complete, and no optimal solutions can be found in reasonable time. Instead, several methods have been proposed to find acceptable and near optimal solutions.

The methods that have been proposed to solve the data mapping problem fall into two major categories: heuristics and physical optimization algorithms. Heuristic procedures are mainly geometry based methods. The physical optimization algorithms are derived from natural sciences. Heuristics include recursive coordinate bijection, recursive spectral bijection, recursive graph bijection, mincut-based, scattered decomposition, nearest-neighbor techniques, greedy algorithms, and clustering methods [Berger and Bokhri 1987; Chrischoides et al. 1991; Ercal 1988; Fox 1988; Karmer and Muhlenbein 1989]. On the other hand, physical optimization algorithms include neural networks [Hopfield and Tank 1986; Fox and Furmanski 1988], simulated annealing [Kirkpatrick et al. 1983; Rutenbar 1989], genetic algorithms [Holland 1975; Goldberg 1989].

Heuristics for data mapping are fast. However, most of them are biased towards certain problem structures and multicomputer topologies. Most heuristics are not guided by an objective function the fact that leads to an unbalanced treatment of the computation and communication terms. Physical optimization algorithms, particularly genetic algorithms

and simulated annealing, are slow in their execution. However, they do not make apriori assumptions concerning the underlying problem, and they are guided by an objective function.

In this thesis, we have worked on two physical optimization methods to solve the mapping problem: mean field annealing and simulated tempering. We study the effects of user parameters in MFA and recommended appropriate values. Also, we adapt simulated tempering for solving the mapping problem and present comparative experimental results of mean field annealing and simulated tempering with Simulated Annealing, Genetic Algorithm and Neural Network.

This thesis is organized as follows. Chapter 2 defines the data mapping problem. In chapter 3, we present mean field annealing and how it is used in solving the data mapping problem. Chapter 4, outlines the simulated tempering algorithm. In chapter 5, we study the sensitivity of MFA to user parameters. In chapter 6, we present comparative graphs of random test cases. Finally, in chapter 7 we present our conclusions and remarks.

Chapter 2

Data Mapping Problem

1. Introduction

Wide use of parallel computers in various computation intensive applications makes the problem of mapping parallel programs to parallel computers more crucial. The mapping problem arises as parallel programs are developed for distributed-memory, message passing parallel computers, which are usually called multicomputers. Processors of a multicomputer are usually connected by utilizing one of the well-known direct interconnection network topologies such as ring, mesh, or hypercube. Designing efficient parallel algorithms for such architectures is not straightforward. An efficient parallel algorithm should exploit the full potential power of the architecture. Processor idle time and the interprocessor communication overhead may lead to poor utilization of the architecture, hence poor overall system performance.

Parallel algorithm design for multicomputers can be divided into two steps. The first step is the decomposition of the problem into a set of interacting sequential sub-problems (or tasks) which can be executed in parallel. The second step is mapping each of these tasks to

an individual processor of the parallel architecture in such a way that the total execution time is minimized. The second step, called the mapping problem, is crucial in designing efficient parallel programs. In general the mapping problem is known to be NP-hard[Indurkhya et al. 1986; Kasahara and Narita 1984]. Hence heuristics giving suboptimal solutions are used to solve the problem. Heuristics proposed to solve the mapping problem are computation intensive. Solving the mapping problem can be considered as a preprocessing performed before the execution of the parallel program on the parallel computer. Sequential execution of the mapping heuritic may introduce unacceptable preprocessing overhead, limiting the efficiency of the parallel implementation. Efficient parallel mapping heuristics are needed in such cases.

2. The Mapping Problem

Classes of problems that have static interaction pattern among the tasks can be represented by a static task graph. The vertices of this graph represent the atomic tasks and the edges represent the interaction among the tasks. Computational costs of the tasks can be estimated prior to the execution of the parallel program. Hence, weights can be associated with the vertices in order to denote the computational costs of the corresponding tasks. In this work, we model the static task interaction pattern by two graphs : Task Precedence Graph (TPG) and the Task Interaction Graph (TIG). TPG is a directed graph where directed edges represent execution dependencies. TIG represents interaction by undirected edges between vertices. In this model, each atomic task can be

executed simultaneously and independently. Edges may be associated with weights which denote the amount of bidirectional information exchange involved between pairs of tasks. Parallel architecture must also be modeled in a way that represents its architectural features. Parallel architectures can easily be represented by a Processor Organization Graph (POG), where nodes represent the processors and edges represent the communication links. Nodes and edges of a POG are not associated with weights since most of the commercially available multicomputer architectures are homogeneous with identical processors and communication links. The communication topology of the multicomputer can be modeled by an undirected complete graph, referred here as the Processor Communication Graph (PCG). The nodes of the PCG represent the processors and the weights associated with the edges represent the unit communication costs between pairs of processors. The PCG can easily be constructed using the topological properties of POG and the routing scheme utilized for the interprocessor communication.

The objective in mapping TIG to PCG is the minimization of the expected execution time of the parallel program on the target architecture. Thus, the mapping problem can be modeled as an optimization problem by associating the following quality measures with a good mapping: (i)interprocessor communication overhead should be minimized: (ii)computational load should be uniformly distributed among processors in order to minimize processor idle time.

A mapping problem instance can be represented with two undirected graphs, the Task Interaction Graph (TIG) and the Processor Communication Graph (PCG). The vertices of the TIG represent the tasks that are to be mapped. Each vertex is associated with a weight that represents the computational *cost* of the task. Edges of the TIG represent the communication required among the tasks. Also these edges are associated with weights denoting the *volume* of interaction required between the two tasks connected by each edge. The vertices of the PCG represent the processors of the target multicomputer. The edge weights in the PCG, represent the unit communication cost between the processors. The problem that we are trying to solve is mapping the tasks of TIG to the processors in PCG.

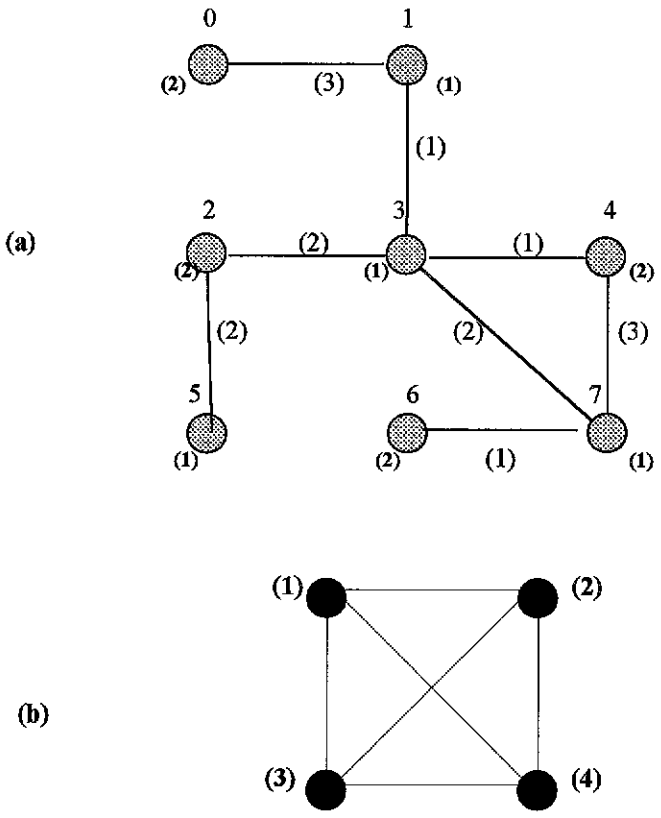


Figure 1. A mapping instance, (a) represents the TIG, (b) PCG of 2-dimensional hypercube

The question is to find a many-to-one mapping function which assigns each vertex of the TIG to a unique node of the PCG, and minimizes the total interprocessor communication cost, while maintaining the computational load of each processor balanced. The interprocessor communication cost is equal to the product of the volume of interaction between two tasks and the unit communication cost between the processors to which each of these tasks is mapped. The computational load of a processor is equal to the summation of the weights of the tasks assigned to that processor. Perfect load balance is achieved by assigning to each processor equal load.

3. Data Mapping Solution Efficiency

The speed up (S) of the multicomputer is defined as being the workload of a sequential processor divided by the workload of the slowest processor in the multicomputer (the value of the objective function after finding the mapping solution represents the workload of the slowest processor).

$$S = \text{(Workload of a single processor)} / \text{(Workload of the slowest processor in multicomputer)}$$

The solution quality (η) is determined by the efficiency resulting from the mapping solutions. It is defined as the speed up (S) divided by the number of processors involved.

Chapter 3

Mean Field Annealing for Data Mapping

1. Mean Field Annealing

Mean Field Annealing (MFA) merges the collective computation and annealing properties of Hopfield Neural Network (HNN) [Hopfield and Tank 1985, 1986, 1987] and Simulated Annealing (SA) [Kirkpatrick et al. 1983], respectively, to obtain a general algorithm for solving combinatorial optimization problems. A major drawback of HNN is that it does not have a good scaling property, which is a very important performance criterion for heuristic optimization algorithms. MFA has been proposed as a successful alternative [Peterson and Soderberg 1989; Van den Bout and Miller 1988, 1989, 1990]. In the MFA algorithm, the problem representation is identical to HNN [Hopfield and Tank 1985; Van den Bout and Miller 1988, 1989], but the iterative scheme used to relax the system is different. MFA can be used to solve a combinatorial optimization problem by choosing a representation scheme in which the final states of the *spins* can be decoded as a solution to the target problem.

The MFA algorithm is derived by making an analogy to the Ising spin model which is used to estimate the state of a system of particles or spins in thermal equilibrium.. In the Ising spin model, the energy of a system with S spins has the following form :

$$H(s) = \frac{1}{2}\sum_{k=1}^{S}\sum_{l\neq k}\beta_{kl}s_k s_l + \sum_{k=1}^{S}h_k s_k \qquad (1)$$

Here, β_{kl} indicates the level of interaction between spins k and l, and $s_k \in \{0,1\}$ is the value of spin k. At the thermal equilibrium, spin average $\langle s_k \rangle$ of spin k can be calculated using Boltzmann distribution as follows [Van den Bout and Miller 1988]

$$\langle s_k \rangle = \frac{1}{1+e^{-\phi_k/T}} \qquad (2)$$

Here, $\phi_k = \langle H(s) \rangle\big|_{s_k=0} - \langle H(s) \rangle\big|_{s_k=1}$ represents the *mean field* acting on spin k, where the energy average $\langle H(s) \rangle$ of the system is

$$\langle H(s) \rangle = \sum_{k=1}^{S}\sum_{l\neq k}\beta_{kl}\langle s_k s_l \rangle + \sum_{k=1}^{S}h_k \langle s_k \rangle \qquad (3)$$

The complexity of computing ϕ_k using Equation (3) is exponential [Van den Bout and Miller 1988]. However, for large numbers of spins, the *mean field approximation* can be used to compute the energy average as

$$\langle H(s)\rangle = \sum_{k=1}^{S}\sum_{l\neq k}\beta_{kl}\langle s_k\rangle\langle s_l\rangle + \sum_{k=l}^{S} h_k\langle s_k\rangle \qquad (4)$$

Since $\langle H(s)\rangle$ is linear in $\langle s_k\rangle$, the mean field ϕ_k can be computed using the equation

$$\phi_k = \langle H(s)\rangle\Big|_{s_k=0} - \langle H(s)\rangle\Big|_{s_k=1} = -\frac{\partial\langle H(s)\rangle}{\partial\langle s_i\rangle}$$

$$= -\left(\sum_{l\neq k}\beta_{kl}\langle s_l\rangle + h_k\right) \qquad (5)$$

This will reduce the complexity of computing ϕ_k to O(S).

At each temperature, starting with initial spin averages, the mean field acting on a randomly selected spin is computed using Equation (5). Then the spin average is updated using Equation (2). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. The general form of the MFA algorithm derived from this iterative relaxation scheme is shown in Figure 1. The MFA algorithm is

used to find the equilibrium point of a system of S spins using an annealing process similar to Simulated Annealing.

1. Get the initial temperature T_0, and set $T = T_0$

2. Initialize the spin averages $\langle s \rangle = [\langle s_1 \rangle, \langle s_2 \rangle, \ldots, \langle s_k \rangle, \ldots, \langle s_s \rangle]$

3. **While** temperature T is in the cooling range **DO**

    3.1 **While** system is not stabilized for current temperature **DO**

        3.1.1 Select a spin k at random

        3.1.2 Compute $\phi_k, \phi_k = -\sum_{l \neq k} \beta_{kl} \langle s_l \rangle - h_k$

        3.1.3 Update $\langle s_k \rangle, \langle s_k \rangle = \left\{ 1 + e^{-\phi_k / T} \right\}^{-1}$

    3.2 Update T according to the cooling schedule

Figure 1. The Mean Field Annealing algorithm

2. MFA for Data Mapping

In this section, we present Bultan and Aykanat's formulation [Bultan and Aykanat 1992] of the MFA algorithm for the mapping problem and its pseudo code which is given in Figure 2.

A spin matrix, which consists of N (task) rows and K (processor) columns, is used as the representation scheme. That is, N × K spins are used to encode the solution. The output S_{ip} of a spin (i, p) denotes the probability of mapping task i to processor p. The spins are

continuous variables in the range $0 \le s_{ip} \le 1$. When the MFA algorithm reaches a solution, the spin values converge to either 0 or 1 indicating the result. If s_{ip} converges to 1, this means that task i is mapped to processor p. For example, consider a mapping problem of eight tasks and four processors, a solution for the problem will be presented by 8×4 spin matrix :

		4 Processors			
		1	2	3	4
8 Tasks	1	0	0	0	1
	2	0	0	0	1
	3	0	1	0	0
	4	1	0	0	0
	5	0	0	1	0
	6	0	1	0	0
	7	1	0	0	0
	8	0	0	1	0

The following energy function is proposed for the mapping problem :

$$H(s) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}\sum_{p=1}^{K}\sum_{q\neq p} e_{ij}\, s_{ip}\, s_{jq}\, d_{pq} + \frac{r}{2}\sum_{i=1}^{N}\sum_{j\neq i}\sum_{p=1}^{K} s_{ip}\, s_{jp}\, w_i\, w_j \qquad (6)$$

Here e_{ij} denotes the edge weight between the pair of tasks i and j, and w_i denotes the weight of task i in TIG. The edge weight between processors p and q in PCG is represented by d_{pq}. The term $s_{ip} \times s_{jq}$ denotes the probability that task i and task j are mapped to two different processors p and q, respectively. Hence, the term $e_{ij} \times s_{ip} \times s_{jq} \times d_{pq}$ represents the weighted interprocessor communication overhead

introduced due to the mapping of tasks i and j to different processors. Our aim is to minimize the first quadruple summation term which corresponds to the minimization of the interprocessor communication overhead.

The second triple summation term computes the summation of the inner products of the weights of the tasks mapped to individual processors. The global minimum of this term occurs when equal amounts of task weights are mapped to all processors.

The parameter r is introduced to maintain a balance between the two optimization objectives of the mapping problem.

The expression for the mean field ϕ_{ip} experienced by spin(i, p) is

$$\phi_{ip} = - \frac{\delta H(s)}{\delta s_{ip}} = - \sum_{j \neq i}^{N} \sum_{q \neq p}^{K} e_{ij} s_{jq} d_{pq} - r \sum_{j \neq i}^{N} s_{jp} w_i w_j \qquad (7)$$

In a feasible mapping, each task should be mapped exclusively to a single processor. This constraint is explicitly handled while the spin values are updated. Individual spin value s_{ip} is proportional to $e^{\phi_{ip}/T}$. Then, s_{ip} can be normalized as

$$S_{ip} = \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^{K} e^{\phi_{ip}/T}} \qquad (8)$$

This normalization compels the summation of each row of the spin matrix to be equal to unity.

Equation (7) can be interpreted in the context of the mapping problem as follows. The first double summation term represents the increase in the total interprocessor communication cost by mapping task i to processor p. The second summation term represents the increase in the computational load balance cost associated with processor p by mapping task i to processor p. Hence, $-\phi_{ip}$ may be interpreted as the decrease in the overall solution quality obtained by mapping task i to processor p. Then, in Equation (8), S_{ip} is updated so that the probability of task i being mapped to processor p increases with increasing mean field ϕ_{ip} experienced by spin(i,p).

In the presented formulation of MFA for the mapping problem, K spins of a randomly chosen row of the spin matrix are updated at a time. Mean fields ϕ_{ip} ($1 \le p \le K$) experienced by the spins at the ith row of the spin matrix are computed by using Equation (7) for p = 1, 2, ... K. Then, the spin averages S_{ip}, $1 \le p \le K$ are updated using Equation (8) for p = 1, 2, ... K. Each row update of the spin matrix is referred as a single

iteration of the algorithm. The system is observed after each spin-row update in order to detect the convergence to an equilibrium state for a given temperature [Van den Bout and Miller 1989]. If the energy function H does not decrease after a certain number of consecutive spin-row updates, this means that the system is stabilized for that temperature [Van den Bout and Miller 1989]. Then T is decreased according to a cooling schedule, and the iterative spin update is repeated.

The implementation of MFA is based on [Bultan and Aykanat 1992]. At each temperature, iterations continue until $\Delta H < \varepsilon$ (line 3.1 in Figure 2) for L consecutive iterations, initially L is set equal to N (number of tasks).The parameter ε is chosen to be 0.5. The cooling process is realized in two phases, slow cooling followed by fast cooling. In the slow cooling phase, temperature is decreased using α=0.9 until T is less than $T_o/1.5$. Then, in the fast cooling phase, L is set to L/4, α is set to 0.5 and cooling is continued until T is less than $T_o/5$. At the end of this cooling process, maximum spin value at each row is set to 1 and all other spin values are set to zero.

The coefficient r, is computed after the spins are initialized randomly using the following formula :

$$r = \left(\sum_{i=1}^{N} \sum_{j \neq i} \sum_{p=1}^{K} \sum_{q \neq p} e_{ij} s_{ip} s_{jq} d_{pq}\right) / K * \left(\sum_{i=1}^{N} \sum_{j \neq i} \sum_{p=1}^{K} s_{ip} s_{jp} w_i w_j\right)$$

1. Get the initial temperature T_0, and set $T = T_0$
2. Initialize the spin averages $s = \{ s_{11}, \ldots, s_{ip}, \ldots, s_{NK} \}$
3. **While** temperature T is in the cooling range **DO**
   - 3.1 **While** H is decreasing **DO**
     - 3.1.1 Select a task i at random
     - 3.1.2 Compute mean fields of the spins at the i-th row

$$\phi_{ip} = -\sum_{j \neq i}^{N} \sum_{q \neq p}^{K} e_{ij} s_{jq} d_{pq} - r \sum_{j \neq i}^{N} s_{jp} w_i w_j \quad for\ 1 \leq p \leq K$$

     - 3.1.3 Compute the summation $\sum_{p=1}^{K} e^{\phi_{ip}/T}$
     - 3.1.4 Compute new spin values at the i-th row

$$s_{ip}^{new} = e^{\phi_{ip}/T} \Big/ \sum_{p=1}^{K} e^{\phi_{ip}/T} \quad for\ 1 \leq p \leq K$$

     - 3.1.5 Compute the energy change due to these spin updates

$$\Delta H = \sum_{p=1}^{K} \phi_{ip} (s_{ip}^{new} - s_{ip})$$

     - 3.1.6 Update the spin values at the i-th row

$$s_{ip} = s_{ip}^{new} \quad for\ 1 \leq p \leq K$$

   - 3.2 $T = \alpha \times T$

Figure 2. Mean Field Annealing algorithm for the mapping problem.

Chapter 4

Simulated Tempering for Data Mapping

1. Simulated Tempering

Simulated Tempering was proposed by Parisi and Marinari [Marinari and Parisi 1992] to study Random Field Ising Model. These models have a first order phase transition, so there are two coexisting states at T_c, a high energy and low energy state. Standard Metropolis algorithm tends to get stuck in one of these states. The basic idea of Simulated Tempering is try to change the temperature stochastically so that the system can remain in equilibrium when changing the temperature. In this way, the high energy and low energy states will be visited accordingly when we study systems with rough free energy landscape. The main difficulty of Simulated Annealing as an optimization method is that, in order to keep the process in quasi-equilibrium, the temperature decrements must be chosen such that the steps do not disturb the equilibrium density too much, and the system must be thermalized before decreasing the temperature. Consequently this class of algorithms will be very time consuming. In contrast to Simulated Annealing, the

Tempering method will keep the system in equilibrium when changing the temperature so that there are no time lost in thermalizing at each temperature. Secondly, the decrements of the temperature can be chosen more freely as long as the acceptance rates of updating the temperature stay in a reasonable range. Thirdly, Simulated Tempering heats and cools the system constantly, allowing it to explore many local minima.

To achieve the goals stated above we now present the basic properties of Simulated Tempering. Let X be the configuration space. Enlarge the space by adding a new variable m, which can take M values(m=1..M). The probability distribution p(X,m) is chosen to be $p(X,m) \cong e^{-H(X,m)}$ where the temperature factor β is absorbed in the definition of the Hamiltonian

$$H(X,m) \equiv \beta_m H(X) - g_m \qquad (1)$$

It is evident that the probability induced by the Hamiltonian Equation 1, restricted to the subspace at a fixed m, is the usual Boltzmann distribtion for $\beta=\beta_m$. On the other hand the probability of having a given value of m is

$$p_m \cong Z_m e^{g_m} \equiv e^{-(\beta_m f_m - g_g)} \qquad (2)$$

If we make the choice

$$g_m = \beta_m f_m = - \ln Z_m \quad (3)$$

where Z_m is the partition function at given β_m, then all the p_m's become equal to 1/M.

The transition from one β_m to another must happen with non-negligible probability. Let us try to compute the probability for going from β_m to $\beta_{m+1} \equiv \beta_m + \delta$. If we try to modify β, the variation of the Hamiltonian is given by

$$\Delta H = E\delta - (g_{m+1} - g_m) \quad (4)$$

where E is the instantaneous value of the energy H(X). On the other hand we have that $g_{m+1} - g_m$ is given by the value of the energy for some β in between β_m and β_{m+1}. More precisely

$$g_{m+1} - g_m = E_m\delta + 1/2 C_m\delta^2 + O(\delta^3) \quad (5)$$

where E_m is the average energy and C_m is equal to $dE_m/d\beta$. Δg_m is in fact dependent on the average energy at a temperature between β_m and β_{m+1}. If we assume that E is very close to E_m the variation ΔH will not be too large under the condition that

$$C_m\delta^2 = O(1) \quad (6)$$

This condition is equivalent to requiring that there is a non-negligible overlap in the values of the energy computed at contiguous values of temperature. The main difficulty in this method is the required tuning in the choice of g_m. This will be discussed in the next section.

2. Algorithm and Implementation Issues

In this section, we first present a pseudo code of the Tempering algorithm. The input to the program is the temperature schedule with Mβ values in ascending order and the corresponding Δg_m's.

1. Do a number of Metropolis update to thermalize the system at β_1.

2.Let β_m be the current temperature.Change to the neighboring β's in the following way :

a) Generate a random number y between 0 and 1.

b) If y < 0.5, try to change the temperature to β_{m-1}, otherwise attempt β_{m+1}.

c) Accept the temperature change with probability $e^{-\Delta H(+)}$, where $\Delta H(+)$ is the positive part of ΔH Equation (4).

3. Do a number of Metropolis update on the current temperature.

4. Loop step 2 and 3 until no improvement for N times.

Figure 1. Simulated Tempering algorithm for Data Mapping

The temperature schedule and the corresponding Δg_m's can be obtained in two ways. We can accept rough estimates of the parameters resulted from a preliminary Annealing run and adjust them while running the Tempering algorithm, or we can acquire fairly good approximates using the Annealing algorithm and keep the parameters intact during the Tempering run. In both ways we need to run the Annealing algorithm to get the input to the Tempering algorithm. From the discussions in [Marinari and Parisi 1992] the static method can actually get reasonable results and is simpler to implement than the dynamic method.

One of the main difficulties in the implementation is to find an efficient way to estimate g_m such that the acceptance rate of jumping from one temperature to another will be approximately the same as that of jumping reversely. To get acceptable estimates of Δg_m we can partition the interval δ_m into n equal subintervals and calculate the average energy at each temperature, then approximate Δg_m by the following formula :

$$\Delta g_m = \delta_m / n \left[\left(\sum_{i=1}^{n} E_{i,m} \right) - \frac{1}{2} \left(E_{1,m} + E_{n,m} \right) \right] \qquad (7)$$

where $E_{1,m} = E_m$, $E_{n,m} = E_{m+1}$

3. Objective Function

The objective function used in our implementation is from [Mansour 92]. The computation graph is $G_c(V_c,E_c)$ where V_c represent the task nodes and E_c the communication edges. The multiprocessor is represented by the graph $G_m(V_m,E_m)$ where V_m represent the processor nodes and E_m physical interconnection between the nodes.

OF_{typ} is a typical function corresponding to the time taken by the slowest processor in computing and communicating in a loosely synchronous computation model

$$OF_{typ} = \mathrm{Max}_{p \in Vm}\{W(p)+C(p)\} \quad \textbf{(8)}$$

$W(p)$ corresponds to the computation workload of processor p. It is given by

$$W(p)= \sum_{v \in Vc} w(v).\delta(v,p) \quad \textbf{(9)}$$

where

$$\delta(v,p)=\begin{cases}1 & \textit{if } v \textit{ is mapped to } p \\ 0 & \textit{otherwise}\end{cases}$$

$w(v)$ is the computation time per vertex v. It is given by

$$w(v)=t_{float}.\lambda.\theta(v) \quad \textbf{(10)}$$

where t_{float} is the time taken by a processor to perform a floating point operation. Hence, it is the smallest reasonable time of a machine operation. λ corresponds to the number of computation operations per an edge in E_c in one iteration. $\theta(v)$ is the degree of vertex v in G_c. Substituting Equation (10) in Equation (9) yields

$$W(p)= t_{float}\lambda.\sum_{v\in Vc} \theta(v).\delta(v,p) \quad \textbf{(11)}$$

Let $S_v(p)= \sum_{v\in Vc} \theta(v).\delta(v,p)$. That is, $S_v(p)$ denotes the number of edges in E_c which are mapped to processor p. Thus,

$$W(p)= t_{float}\lambda.S_v(p) \quad \textbf{(12)}$$

$C(p)$ in Equation (8) corresponds to the amount of communication processor p is performing. Clearly $C(p)$ depends on the multicomputer specifications, and hence it will differ from one machine to another. An expression for $C(p)$ which is suitable for modern multicomputers is given in Equation (13). It depends on the message latency and the number of processors a given processor communicates with. Recall that the processor

nodes in a hypercube are numbered in such a way that the Hamming distance of two nodes gives the number of hops between them.

$$C(p)= t_{float}. \sum_{v \in Vc} [\rho.B(p,q)+\sigma+\tau.H(p,q)].\mathrm{sgn}(B(p,q)) \quad \textbf{(13)}$$

ρ is the machine time needed to communicate one word. It is given by the number of "t_{float}"s (i.e., it is divided by t_{float}) required to communicate a word. σ is the message starting time divided by t_{float}. τ is the number of "t_{float}"s representing the communication time per unit distance. Clearly, ρ, σ and τ are machine dependent parameters. $B(p,q)$ is the number of vertices mapped to p and are boundary with q. It is given by

$$B(p,q)=b. \sum_{v \in Vc} \delta(v,p). \delta(u,q) \quad \textbf{(14)}$$

b represents the number of values to be communicated per vertex. $H(p,q)$ in Equation (13) corresponds to the Hamming distance between processors p and q. Finally

$$\mathrm{sgn}(x)= \begin{cases} 1, x > 0 \\ 0, x = 0 \end{cases}$$

$C(p,q)$ is considered to be a reasonable estimation. It is not precise, but it is popular in the mapping literature. Precise measures are sometimes impossible to express accurately. For

example, link contention, communication-computation overlaps, and synchronization delays are hard to quantify.

Minimizing OF_{typ} is computationally expensive because W(p) and C(p) correspond to conflicting requirements. In other words, it gives rise to a min-max criterion. Maximum workload per processor corresponds to minimum communication and vice versa. Nevertheless, OF_{typ} is not smooth, which means that the computation of an incremental change resulted from the remapping of a vertex from one processor to another is expensive because it may require the calculation of the loads of all processors. We will circumvent this problem by using a quadratice objective function, OF_{appr}, which is considered to be a good approximation of OF_{typ}[Mansour and Fox 1992].

$$OF_{appr} = \lambda^2 . \sum_{p \in Vm} S^2_v(p) + \mu . \sum_{p \in Vm} C(p) \qquad \textbf{(15)}$$

μ represents the relative importance of the communication term to the computation term. It is given by

$$\mu = \mu_{user} \frac{\lambda^2 . \sum \theta(v)}{4 . \rho |V_m| \sqrt{V_c} (\sqrt{|V_m|} - 1) . E_b\_V_b} \qquad \textbf{(16)}$$

The derivation of μ is given in [Mansour 1992]. μ_{user} is a scalar value between 0 and 1, which should be selected by the user based on experience. $E_b\_V_b$ is also a user-defined value which reflects the average ratio of boundary edges to boundary vertices in a mapping solution.

Smooth objective functions are preferred in optimization methods. Moreover, OF_{appr} possesses a locality property which makes the calculation of an incremental change due to remapping of vertices from one processor to another inexpensive. This locality implies that only the processors involved in the remapping and the remapped objects determine the cost. Equation (17) presents ΔOF_{appr} which corresponds to the incremental change resulted from the remapping of v from processor $p1$ to processor $p2$.

$$\Delta OF_{appr}=2.\ \lambda^2.\ \theta(v)[\ \theta(v)+S_v(p2)-\ S_v(p1)]+\ \mu[\Delta C] \qquad \textbf{(17)}$$

Chapter 5

Sensitivity of MFA to User Parameters

In chapter 3, we presented MFA which aims at minimizing the execution time of parallel algorithms on multicomputers.

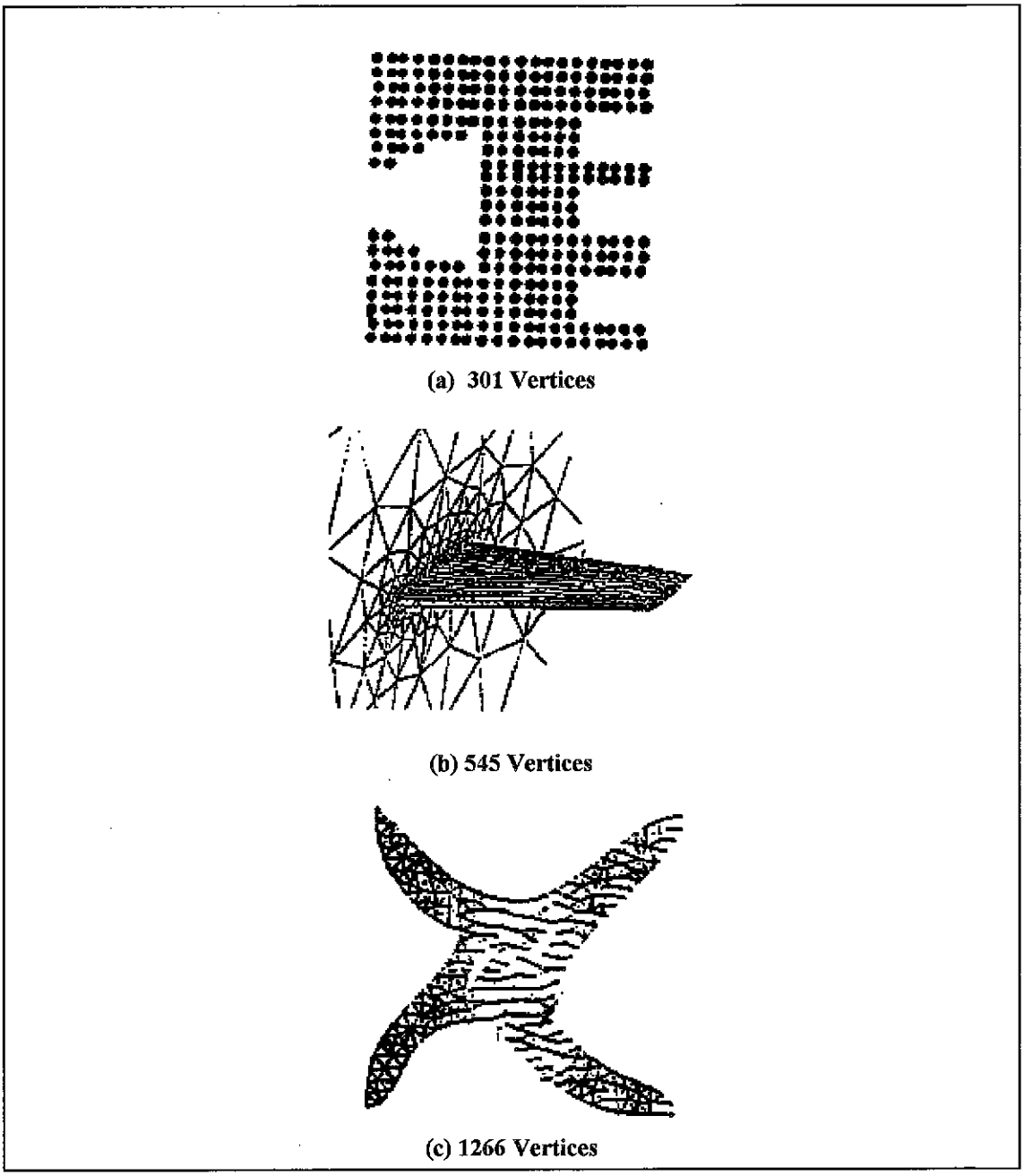
In this chapter we conduct an experimental study to estimate the effects of user intervention when fixing user parameters. The study is done on two test cases: TCASE2 and TCASE3, Figure 1. The behavior of MFA is studied in terms of two measures. The first is solution quality η. The second is MFA's execution time, t_{exec}, measured in seconds. We note that TCASE2 and TCASE3 are contracted once and twice respectively using a graph contraction heuristic [Mansour 1992]. Finally, the experiments are conducted on an AViiON 5000 UNIX machine.

1. Initial Temperature

Figures 2 and 3 show the solution quality and execution time for different initial temperatures. For TCASE2, the solution quality exhibited small fluctuations for low

	Tasks	*Procs*	*Avg. Degree*	*Figure*
TCASE1	301	4	3	Figure 1(a)
TCASE2	545	8	11.5	Figure 1(b)
TCASE3	1266	16	5.5	Figure 1(c)

Table 1. Test cases.



(a) 301 Vertices

(b) 545 Vertices

(c) 1266 Vertices

Figure 1. Shapes of *DATA*. (a) TCASE1. (b) TCASE2. (c) TCASE3.

temperature. As the initial temperature was increased beyond 10, the solution quality started decreasing. For TCASE3, the same performance as for TCASE2, was observed with the decreasing behavior beyond 6.

Execution time, t_{exec}, for both test cases exhibited the same performance. Fluctuations at small temperature and stabilization at high temperature.

From the above results, T_o=5 is an acceptable value and is used during the following experiments.

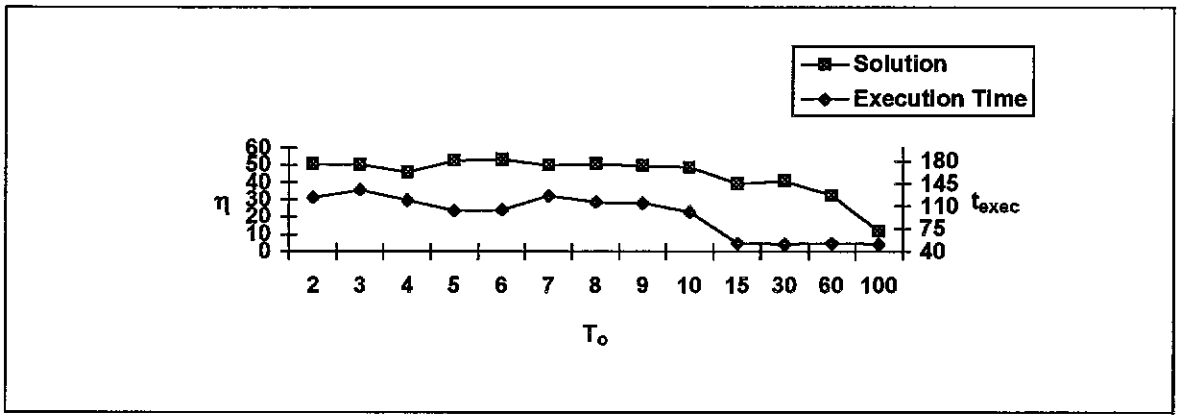


Figure 2. Results of MFA for different initial temperatures using TCASE2.

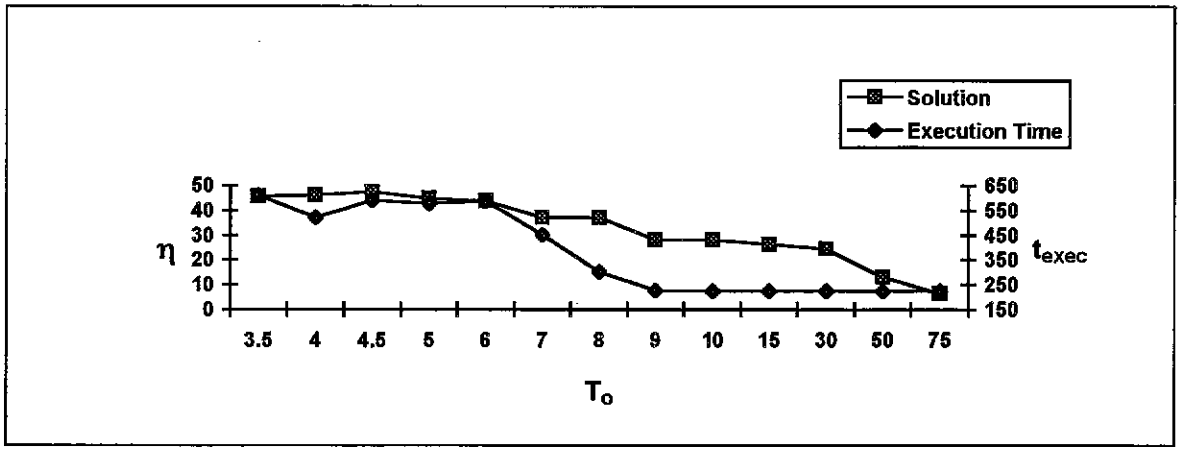


Figure 3. Results of MFA for different initial temperatures using TCASE3.

2. Balancing Coefficient

Figures 4 and 5 show the solution quality and execution time for different balancing coefficients. During the other experiments we used the formula suggested by Bultan and Aykanat. In this experiment we give 'r' the neighboring values of that found by formula. TCASE2 showed an increase in solution quality until it reached at a stable stage (55.8%) beyond r=0.0025. As for TCASE3, the result was not the same. The solution quality did not show any regular behavior and its value varied between 45% and 55%.

As for execution time, for TCASE2, it shows a maximum value of 190secs at 'r'=0.00005. As the value of 'r' is increased, the execution time decreased to less than 75secs. At higher values of 'r', time increased slightly and it showed a near steep behavior. For TCASE3, like solution quality, execution time also showed unstable behavior varying between 340secs at 'r'=0.0006 and 520secs at 'r'=0.002.

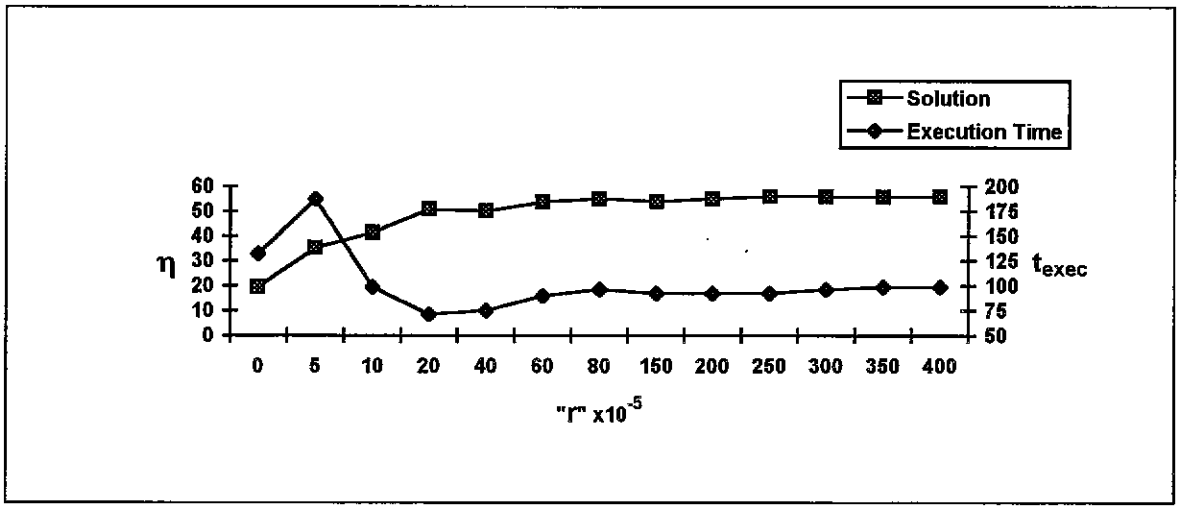


Figure 4. Results of MFA for different balancing coefficients "r" using TCASE2.

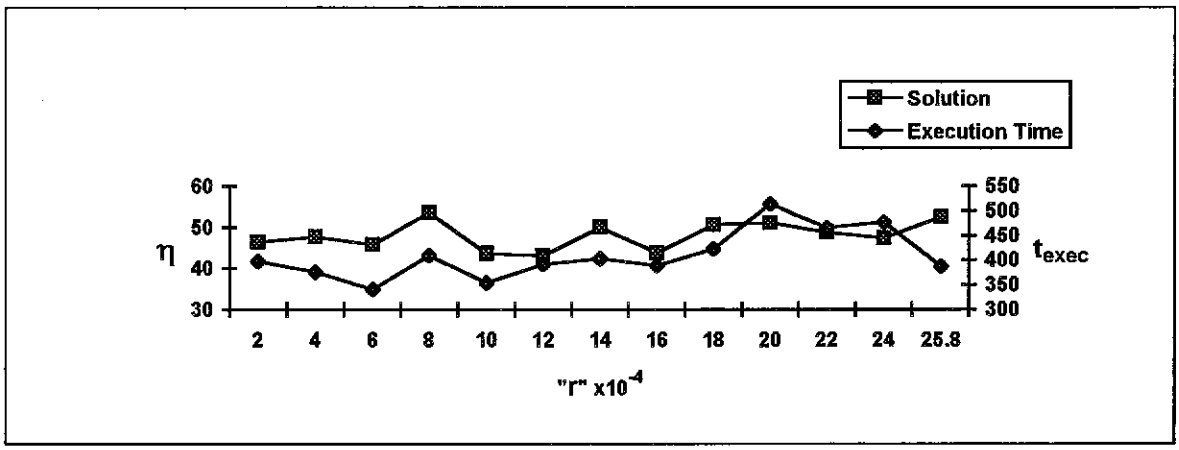


Figure 5. Results of MFA for different balancing coefficients "r" using TCASE3.

3. Cooling Schedule

This parameter is used to handle the cooling schedule (α) of the second phase. Figures 6 and 7 show the solution quality and execution time for different α. For TCASE2 increasing alpha resulted in a decreasing behavior for the solution quality with a maximum value of η=52.8% at α=0.4. For TCASE3, solution quality demonstrated an increasing behavior till α=0.7 η=45.9%. For larger values of alpha solution quality had slight decrease ($\cong$2%).

As for the execution time, it had increasing behavior for both test cases.

For this parameter, we can realize that α=0.5 is an acceptable value for both test cases.

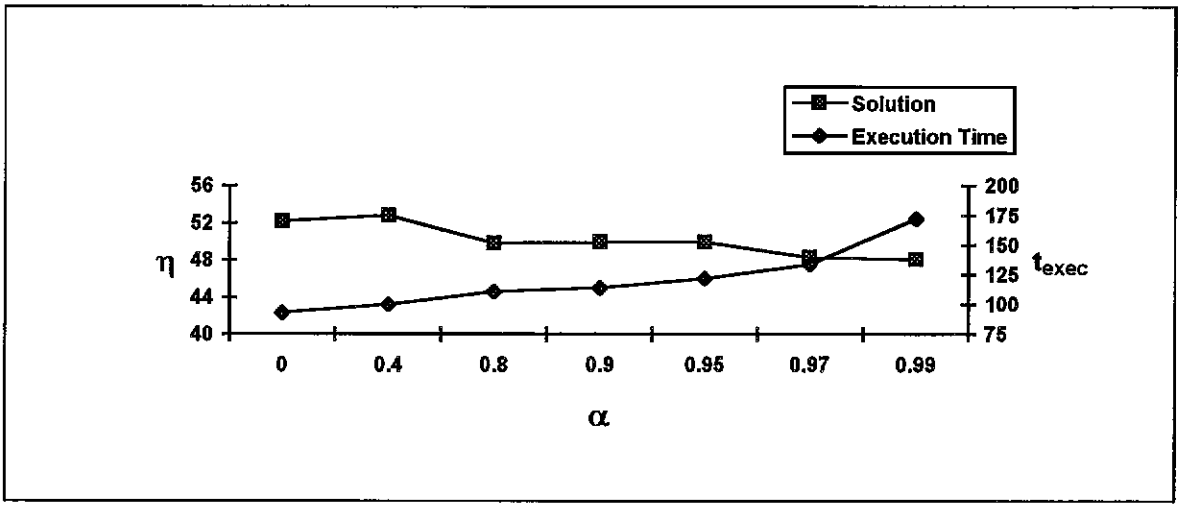


Figure 6. Results of MFA for different cooling schedules using TCASE2.

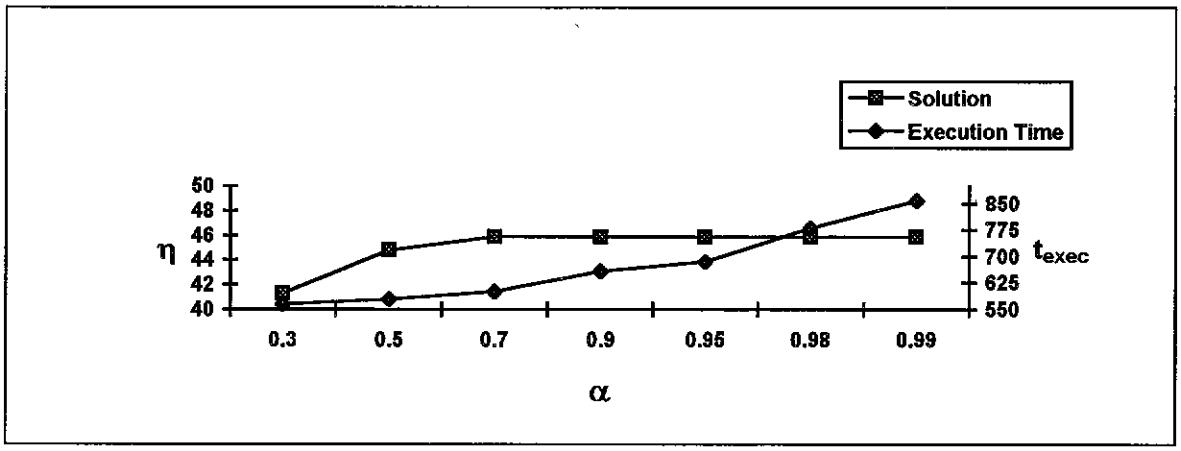


Figure 7. Results of MFA for different cooling schedules using TCASE3.

4. Freezing Temperature

The parameter studied in this experiment, figures 8 and 9, is the freezing temperature in the second cooling phase. Initially, iterations continue until T is less than $T_o/5$. In this experiment we change the value of the divisor. The values on the x-axis are of the divisors.

This parameter behaved differently for each test case. Concerning the solution quality, for TCASE2, it had a maximum value at divisor=5 (53.5%). After this value (divisor=5) it decreased to 49% and remained stable. For TCASE3, solution quality increased reaching 47% and remained stable. As for execution time, for TCASE2 it has parabolic behavior with maximum of 115secs at divisor=15. For TCASE3 it had increasing behavior.

Although there is difference in behavior for each test cases regarding the solution quality but it is observed that value divisor=5 is a good choice.

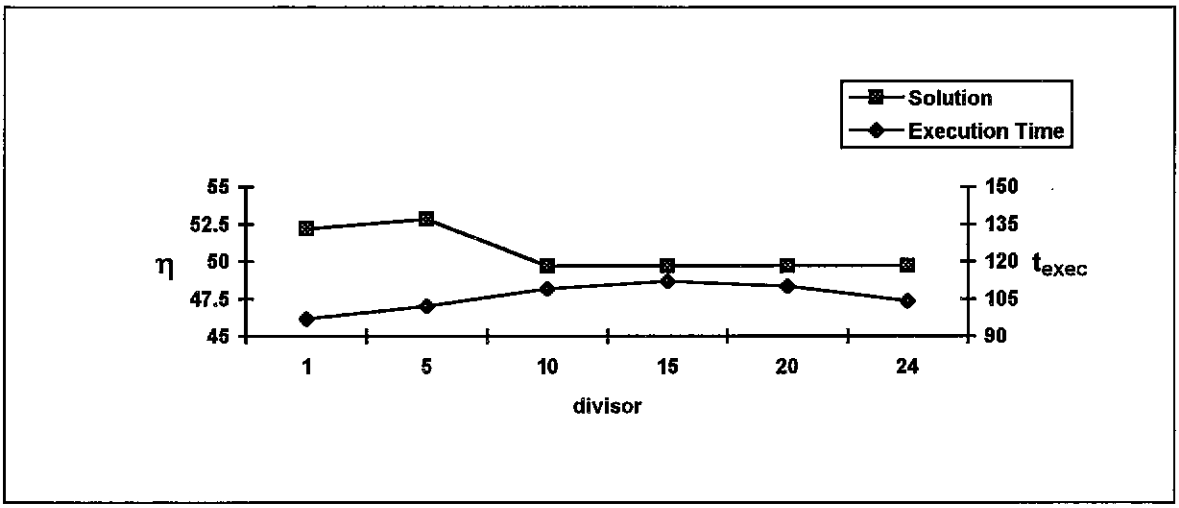


Figure 8. Results of MFA for different freezing temperatures using TCASE2.

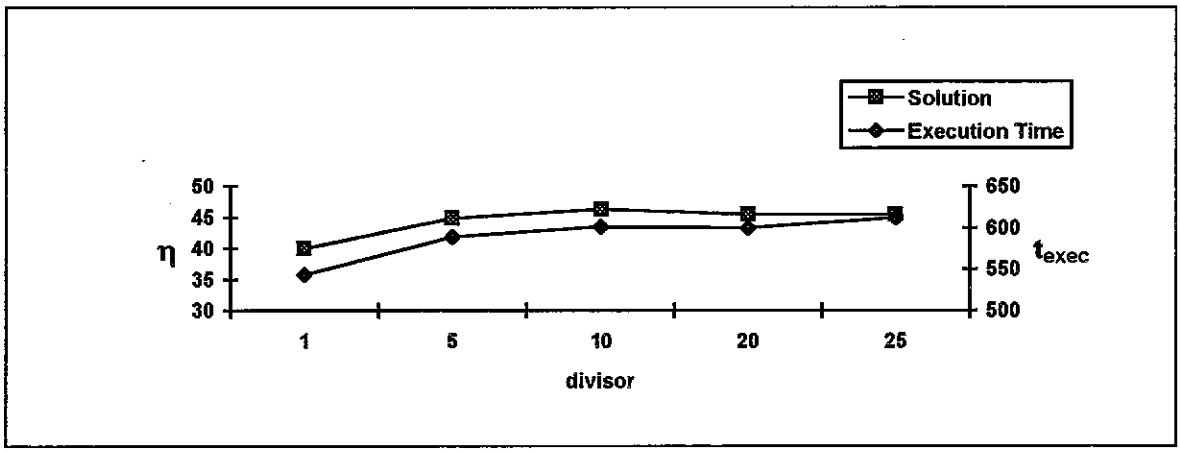


Figure 9. Results of MFA for different freezing temperatures using TCASE3.

5. Convergence Criterion

Figures 10 and 11 show the solution quality and execution time for different convergence criteria(ε). The solution quality, for TCASE2 for values less than 20 had unstable behavior reaching a maximum value of 53%. After epsilon 20, it decreased to minimum value 43% and remained stable. For TCASE3, solution quality started from maximum value (45%), had decreasing behavior between 0.5 and 10, and remained stable after epsilon 10 at a minimum value of 32%.

The execution time had decreasing behavior for both test cases and it stabilized at minimum value beyond ε=20 for TCASE2 and ε=10 for TCASE3.

As for convergence criteria, the value of ε=0.5 is acceptable for both test cases.

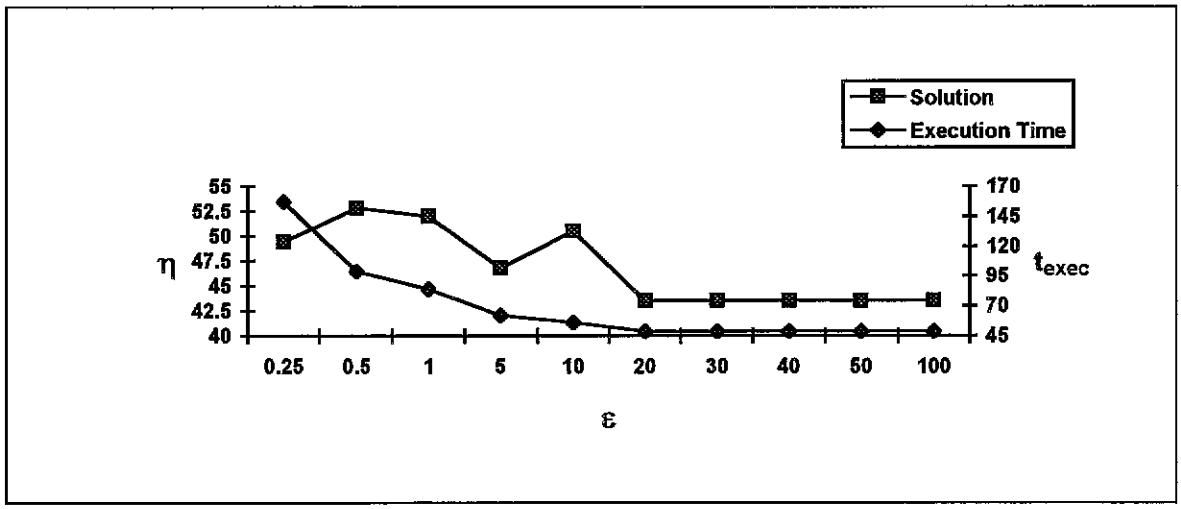


Figure 10. Results of MFA for different convergence criteria using TCASE2.

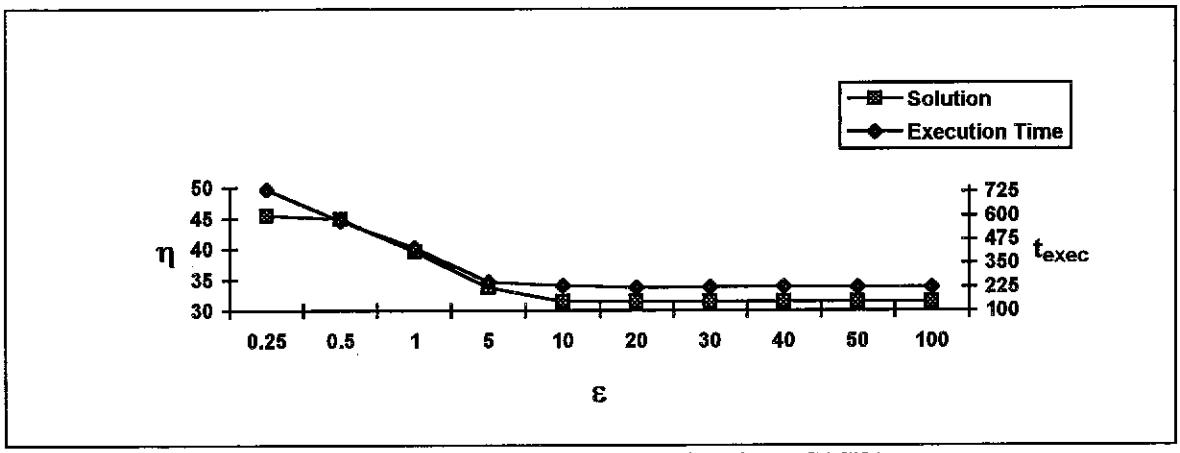


Figure 11. Results of MFA for different convergence criteria using TCASE3.

6. Recommended Parameter Values

We studied in this chapter the sensitivity of MFA to the parameters that require user intervention. We employed in the experimental work two test cases of different characteristics. A summary of the experimental results is given in Table 2.

Parameter	Adopted Value
T_o	5
r	5*r
α	0.5
freezing temp	$T_o/5$
ε	0.5

Table 2. Summary of MFA parameters

Chapter 6

Experimental Results

In this chapter we present the results of comparative study we have done among the following mapping algorithms: Simulated Annealing, Sequential Genetic Algorithm, Neural Network, Mean Field Annealing and Simulated Tempering. We have used the results of SA, NN and SGA reported in [Kawash 94]. In section 1, we present these results for the three test cases TCASE1, TCASE2 and TCASE3. In section 2, we present these results for random graphs generated by random graph generator. In all experiments the MFA uses the adopted values given in Table 2, Chapter 5. Also the execution time of Simulated Tempering excludes the time required to run SA for the temperature and the corresponding Δg_m schedule.

6.1 Results of Different Mapping Algorithms

In Table 1 and Figures 1-6 we present the solution quality (η) and execution time (t_{exec}) of the three test cases, TCASE1 TCASE2 and TCASE3 Table 2, Chapter 5, for the following mapping algorithms: Simulated Annealing (SA), Sequential Genetic Algorithms

(SGA), Neural Network (NN), Mean Field Annealing (MFA) and Simulated Tempering (ST).

	TCASE1		TCASE2		TCASE3	
	η	t_{exec}	η	t_{exec}	η	t_{exec}
SA	65.2%	211	57.2%	1095	54.2%	560
SGA	61.9%	83	59.3%	1652	58.8%	940
NN	60.3%	2	52.3%	5	49.3%	8
MFA	63.7%	47	55.9%	122	53.5%	585
ST	60.3%	17	55.2%	132	50.0%	51

Table 1. Result of different algorithms for TCASE1 TCASE2 & TCASE3

Considering each test case, we can observe that for TCASE1 although the five algorithms gave close solution quality (60.3% NN - 65.2% SA) but this was not the case with the execution time where we observe a low value of 2secs (NN) and a high value of 211 secs (SA). ST and MFA had second and third execution time, respectively. For TCASE2, the solution quality varied between 52.3% (NN) and 59.3% (SGA) but the difference in execution time is considerable (5secs for NN, 1652secs for SGA). MFA had the second best time (122secs) with a solution quality of 55.9%. For TCASE3, the margin is larger for the solution quality than the other test cases (49.3% NN - 58.8% SGA) but still NN has the fastest time (8secs) compared to SGA (940secs).

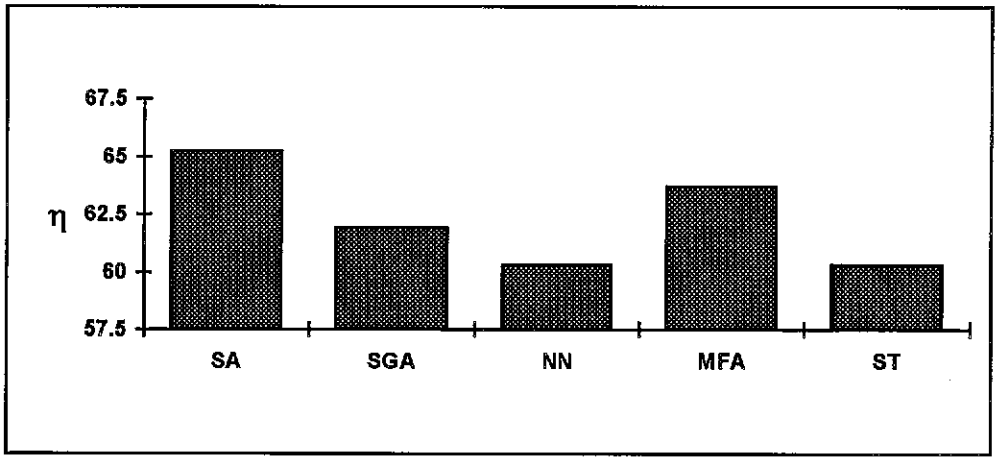


Figure 1. Solution quality for TCASE1.

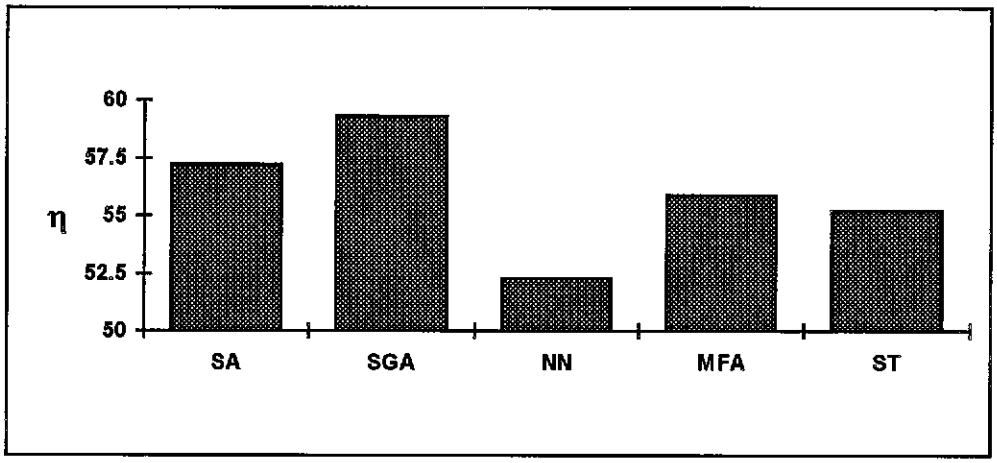


Figure 2. Solution quality for TCASE2.

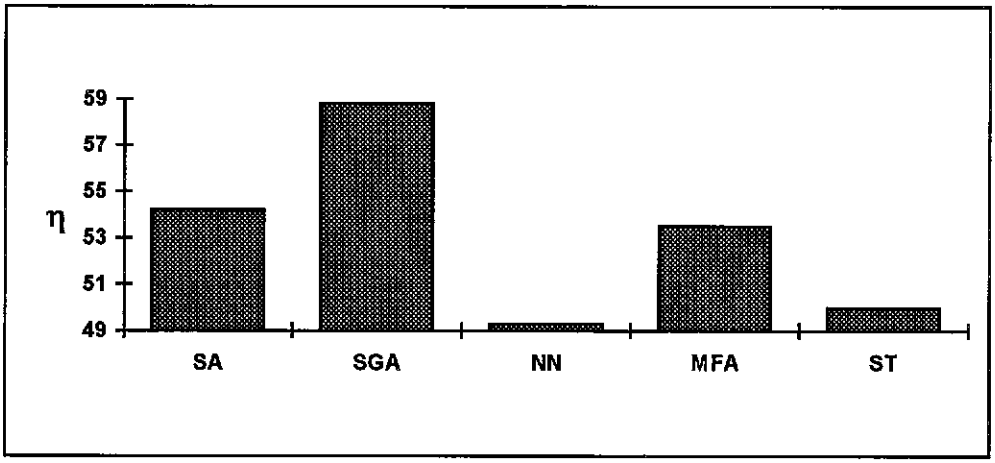


Figure 3. Solution quality for TCASE3.

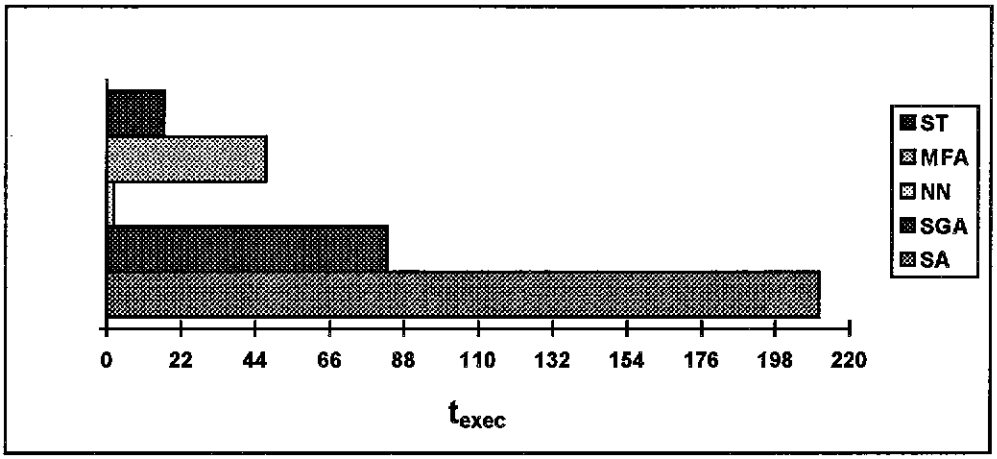


Figure 4. Execution time for TCASE1.

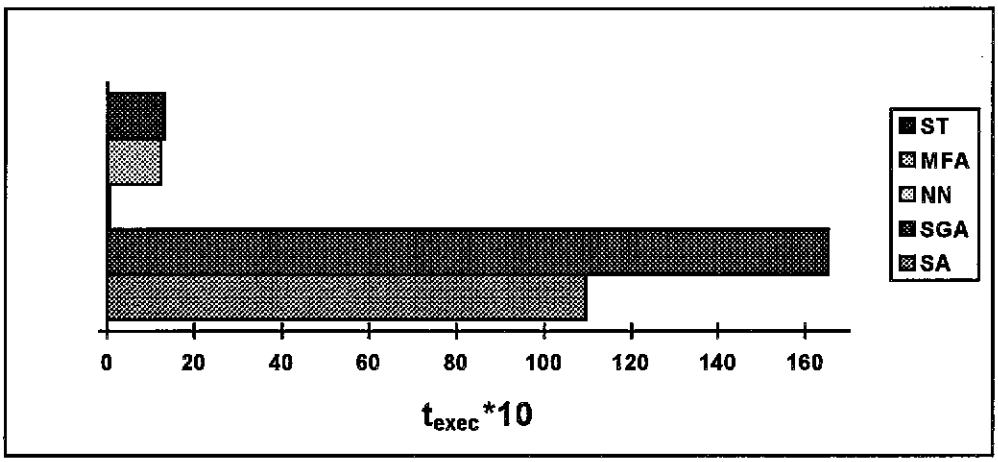


Figure 5. Execution time for TCASE2.

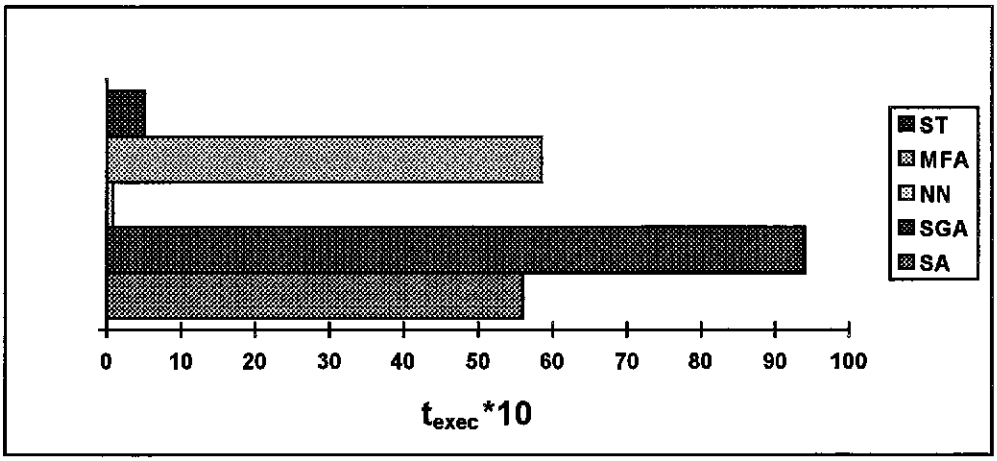


Figure 6. Execution time for TCASE3.

6.2 Results for Random Graph Test Cases

In this section we present the results of 5 selected test cases(SCASEs), generated by random graph generator, mapped to different numbers of processors. The algorithms where executed using the adopted values for the user defined parameters.

The 5 test cases are presented in the following table.

	Vertices	Avg. Degree
SCASE1	300	3
SCASE2	450	3
SCASE3	500	4
SCASE4	750	6
SCASE5	1000	5

Table 2. Randomly Generated Test Cases

The results of the executions are presented in the following tables :

	SA	SGA	NN	MFA	ST
η	45.1%	46.6%	43.5%	43.5%	43.7%
t_{exec}	325	153	2	45	19

Table 3. SCASE1 mapped to 4 processors

	SA	SGA	NN	MFA	ST
η	49.5%	52.6%	47.6%	50.3%	49.3%
t_{exec}	362	722	4	32	23

Table 4. SCASE2 mapped to 4 processors

	SA	SGA	NN	MFA	ST
η	46.2%	53.9%	46.7%	45.7%	43.9%
t_{exec}	589	886	2	18	47

Table 5. SCASE3 mapped to 4 processors

	SA	SGA	NN	MFA	ST
η	27.9%	29.1%	26.2%	21.7%	27.5%
t_{exec}	731	765	4	59	52

Table 6. SCASE3 mapped to 8 processors

	SA	SGA	NN	MFA	ST
η	36.2%	37.5%	32.3%	30.0%	33.3%
t_{exec}	1714	6345	8	100	140

Table 7. SCASE4 mapped to 8 processors

	SA	SGA	NN	MFA	ST
η	33.7%	38.9%	32.1%	38.0%	32.5%
t_{exec}	1713	2717	12	396	131

Table 8. SCASE5 mapped to 8 processors

	SA	SGA	NN	MFA	ST
η	18.0%	18.3%	16.0%	13.6%	17.6%
t_{exec}	2037	2372	17	637	161

Table 9. SCASE5 mapped to 16 processors

The aim of the above tables, is to show the behavior of different algorithms when applied to random graphs of different sizes (and degrees). We can realize that Neural Network (NN) proved to be the fastest algorithm although the quality of its solution is lesser than that of the Sequential Genetic Algorithm (SGA). Simulated Tempering (ST) had the second best time (except in Tables 5 and 7). Genetic Algorithm had the slowest execution but its quality was superior to all other algorithms. Also we note that MFA is most sensitive to an increase in the number of processors.

Chapter 7

Conclusions

In this thesis, we have worked on two physical optimization methods for solving the mapping problem: Mean Field Annealing and Simulated Tempering. We have studied the effects of user defined parameters on MFA's behavior. Our experiments resulted in adopting values to the user parameters based on these test cases. We also implemented a Simulated Tempering algorithm, for the first time, to solve the mapping problem. The implementation was based on the static method where we had to run the Simulated Annealing to obtain the temperature schedule and the corresponding Δg_m's. These values are used as input to run the Simulated Tempering. Further work is required to overrun this problem by accepting rough estimates of these parameters resulted from a preliminary Annealing run and adjust them while running the Tempering algorithm.

We have applied the MFA and ST on randomly generated graphs, together with Simulated Annealing (SA),Sequential Genetic Algorithm (SGA) and Neural Network (NN). The aim of these experiments was to observe the behavior of each algorithm. It was found that Neural Network (NN) proved to be the fastest algorithm although the quality of its

solution is lesser than that of the Sequential Genetic Algorithm, which was superior to all algorithms. As for time, SA had the second slowest time with the second best solution. MFA and ST occupied the third and fourth positions concerning the execution time and solution quality.

Further work includes the implementation of parallel MFA and ST. Also experiments should be conducted on more random graphs inorder to be able to generalize the values adopted in this work.

References

Berger M. and Bokhari S. 1987. A partitioing strategy for nonuniform problems on multicomputers. *IEEE trans. on Parallel and Distributed Systems.* Vol. 1, No. 1, Jan., 91-106.

Brassard G., and Brately P. 1988. *Algorithmics: Theory and Practice.* Printice-Hall International, Inc.

Bultan T., and Aykanat C. 1992. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing,*16, 292-305.

Chrischoides N., Houstis E. N., Houstis C. E. 1991. Geometry based mapping strategies for PDE computations. *Int. Conf. on Supercomputing,* 115-127.

Ercal F. 1988. Heuristic Approaches to Task Allocation for Parallel Computing. Ph.D. Thesis, Ohio State University.

Fox G.C., 1988. A review of automatic load balancing and decomposition methods for the hypercube. *Numerical Algorithms for Modern Parallel Computers,* ed. M. Schultz, 63-76, Sprintg-Verlag.

Fox G.C. and Furmanski, W. 1988. Load Balancing loosely synchronous problems with a neural network. . Proc., 3rd Conf. on Hypercube Concurrent Computers and Applications.

Fox G.C, Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. 1988. *Solving Problems on Concurrent Processors.* Englewood Cliffs, N.J., Printice Hall.

Goldberg D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, Mass., Addison-Wesly.

Holland J. H. 1975. *Adaptation in Natural and Artificial systems.* Univ. of Michigan Press, Ann Arbor.

Hopfield J.J., and Tank D.W. 1985. "Neural computation of decisions in optimization problems". *Biolo. Cybernet.* 52, 141-152.

Hopfield J.J., and Tank D.W. 1986. "Computing with neural circuits: A model", *Science* 233, 625-633.

Hopfield J.J., and Tank D.W. 1987. "Collective computation in neuron like circuits". *Sci. Amer.* 257,6, 104-114.

Hwang K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programability*. McGraw-Hill, Inc.

Indurkhya B., Stone H.S., and Xi-Cheng L. 1986. "Optimal partitioning of randomly generated distributed programs". *IEEE Trans. Software Engineering.* 12,3, 483-495.

Karmer O, and Muhlenbein H. 1989. Mapping Strategies in message-based multiprocessor systems. *Parallel Computing*, 213-225.

Kasahara H., and Narita S. 1984. "Practical Multiprocessor scheduling algorithms for efficient parallel processing". *IEEE Trans. Comput.* 33,11, 1023-1029.

Kawash J. 1994. Sensitivity to Parameters and General Applicability of Genetic Algorithms and Simulated Annealing Algorithms for Mapping Data to Multicomputers. MS thesis.

Kirkpatrick S., Gelatt C.D., and Vecchi M.P. 1983. "Optimization by Simulated Annealing", *Science* 220, 671-680.

Mansour N. 1992. *Physical Optimization algorithms for mapping data to distributed-memory multicomputers*. Ph.D. Diss., Comp. Sc., Syracuse, N.Y.

Mansour N., and Fox G.C. 1992. Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations. *Concurrency: Practice and Experience, 4(7)557-574.*

Marinari E., and Parisi G. 1992. "Simulated Tempering: A New Monte Carlo Technique", Europhys. Lett.

Rutenbar R. A. 1989. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine.*

Peterson C., and Soderberg B. 1989. "A new method for mapping optimization problems onto neural networks". *Int. J. neural Systems* 1,3.

Van den Bout D.E., and Miller T.K. 1988. "A traveling salesman objective function that works". *IEEE Int. Conf. Neural Nets.* Vol. 2, pp. 299-303.

Van den Bout D.E., and Miller T.K. 1989. "Improving the performance of the Hopfield-Tank neural network through normalization and annealing". *Bio. Cybernet* 62, 129-139.

Van den Bout D.E., and Miller T.K. 1990. "Graph partitioning using annealed neural networks". *IEEE Trans. Neural Networks.* 1, 2, 192-203.