

PARALLEL MAPPING AND CIRCUIT
PARTITIONING HEURISTICS BASED ON MEAN
FIELD ANNEALING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Tevfik Bultan

January 1992

QA
402.5
.B85
1992

PARALLEL MAPPING AND CIRCUIT
PARTITIONING HEURISTICS BASED ON MEAN
FIELD ANNEALING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Tevfik Bultan

January 1992

Tevfik Bultan
tarafından bağışlanmıştır.

9A

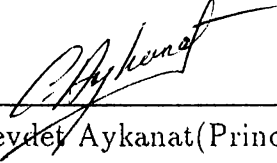
402-5

-B85

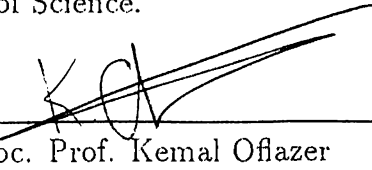
1992

B. 10123

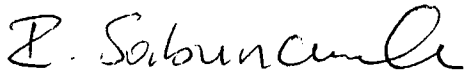
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Assoc. Prof. Ceydet Aykanat (Principal Advisor)


I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Assoc. Prof. Kemal Oflazer

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. İhsan Sabuncuoğlu

Approved by the Institute of Engineering and Science:


Prof. Mehmet Baray, Director of the Institute of Engineering and Science

ABSTRACT

PARALLEL MAPPING AND CIRCUIT PARTITIONING HEURISTICS BASED ON MEAN FIELD ANNEALING

Tevfik Bultan

M. S. in Computer Engineering and Information Science

Supervisor: Assoc. Prof. Cevdet Aykanat

January 1992

Mean Field Annealing (MFA) algorithm, recently proposed for solving combinatorial optimization problems, combines the characteristics of neural networks and simulated annealing. In this thesis, MFA is formulated for the mapping problem and the circuit partitioning problem. Efficient implementation schemes, which decrease the complexity of the proposed algorithms by asymptotical factors, are also given. Performances of the proposed MFA algorithms are evaluated in comparison with two well-known heuristics: simulated annealing and Kernighan-Lin. Results of the experiments indicate that MFA can be used as an alternative heuristic for the mapping problem and the circuit partitioning problem. Inherent parallelism of the MFA is exploited by designing efficient parallel algorithms for the proposed MFA heuristics. Parallel MFA algorithms proposed for solving the circuit partitioning problem are implemented on an iPSC/2¹ hypercube multicomputer. Experimental results show that the proposed heuristics can be efficiently parallelized, which is crucial for algorithms that solve such computationally hard problems.

¹iPSC/2 is a registered trademark of Intel Corporation

Keywords: Mean Field Annealing, Neural Networks, Simulated Annealing, Combinatorial Optimization, Mapping Problem, Circuit Partitioning Problem, Parallel Processing, Multicomputers.

ÖZET

ORTAK ALAN TAVLAMASINA DAYANAN PARALEL EŞLEME VE DEVRE PARÇALAMA ALGORİTMALARI

Tevfik Bultan

Bilgisayar Mühendisliği ve Enformatik Bilimleri Bölümü

Yüksek Lisans

Tez Yöneticisi: Assoc. Prof. Cevdet Aykanat

Ocak 1992

Birleşimsel eniyileme problemlerini çözmek için önerilen Ortak Alan Tavlama (OAT) algoritması, sinir ağları ve tavlama benzetimi yöntemlerinin özelliklerini taşır. Bu çalışmada, OAT algoritması eşleme ve devre parçalama problemlerine uyarlanmıştır. Önerilen algoritmaların karmaşıklığını asimtotik olarak azaltan verimli gerçekleştirme yöntemleri de geliştirilmiştir. Önerilen algoritmaların başarımları tavlama benzetimi ve Kernighan-Lin algoritmaları ile kıyaslanarak değerlendirilmiştir. Elde edilen sonuçlar OAT'nin eşleme ve devre parçalama problemlerini çözmek için alternatif bir algoritma olarak kullanılabileceğini göstermektedir. Önerilen OAT algoritmaları verimli bir şekilde paralelleştirilmiştir. Devre parçalama problemi için önerilen paralel OAT algoritmaları iPSC/2 hiperküp çok işlemcili bilgisayarıda gerçekleştirilmiştir. DeneySEL sonuçlar önerilen algoritmaların verimli bir şekilde paralelleştirilebildiklerini göstermektedir.

Anahtar kelimeler : Ortak Alan Tavlama, Sinir Ağları, Tavlama Benzetimi, Birleşimsel Eniyileme, Eşleme Problemi, Devre Parçalama Problemi, Paralel İşleme, Çok İşlemcili Bilgisayarlar.

ACKNOWLEDGEMENT

I am very grateful to my supervisor Assoc. Prof. Cevdet Aykanat as he taught me what research is, and always provided a motivating support during this study.

I would also like to express my gratitude to Assoc. Prof. Kemal Oflazer and Asst. Prof. İhsan Sabuncuoğlu for their remarks and comments on this thesis.

Finally, I wish to thank all my friends, and my family for their morale support.

Contents

1	INTRODUCTION	1
2	THEORY	6
2.1	Hopfield Neural Networks	6
2.1.1	Combinatorial Optimization Using Hopfield Neural Networks	7
2.1.2	Problems of Hopfield Neural Networks	8
2.2	Simulated Annealing	9
2.3	Mean Field Annealing	11
3	MFA FOR THE MAPPING PROBLEM	14
3.1	The Mapping Problem	14
3.2	Modeling the Mapping Problem	17
3.3	Solving the Mapping Problem Using MFA	21
3.3.1	Formulation	23
3.3.2	An Efficient Implementation Scheme	28
3.4	Performance of Mean Field Annealing Algorithm	30
3.4.1	MFA Implementation	31

3.4.2	Kernighan-Lin Implementation	31
3.4.3	Simulated Annealing Implementation	32
3.4.4	Experimental Results	33
3.5	Parallelization of Mean Field Annealing Algorithm	37
4	MFA FOR THE CIRCUIT PARTITIONING PROBLEM	45
4.1	The Circuit Partitioning Problem	45
4.2	Modeling the Circuit Partitioning Problem	46
4.3	Solving the Circuit Partitioning Problem Using MFA	49
4.3.1	Graph Model	49
4.3.2	Network Model	51
4.4	Parallelization of Mean Field Annealing Algorithm	56
4.4.1	Graph Model	57
4.4.2	Network Model	59
5	CONCLUSIONS	63

List of Figures

2.1	Simulated annealing algorithm.	10
2.2	Mean field annealing algorithm.	12
3.1	A mapping problem instance, with (a) TIG, (b) POG (which represents a 2-dimensional hypercube) and (c) PCG.	22
3.2	MFA algorithm for the mapping problem.	27
3.3	Node program for one iteration of the parallel MFA algorithm for the mapping problem.	43
4.1	Modeling of a given circuit partitioning problem instance with (a) graph and (b) network models. Dashed lines indicate an example partition.	48
4.2	Two possible solutions for the given circuit partitioning problem instance.	54
4.3	Node program for one iteration of the parallel MFA algorithm for the graph partitioning problem.	58
4.4	Speed-up (a) and efficiency (b) curves for the graph partitioning problem.	60
4.5	Node program for one iteration of the parallel MFA algorithm for the network partitioning problem.	61

4.6	Speed-up (a) and efficiency (b) curves for the network partitioning problem.	62
-----	--	----

List of Tables

3.1	Averages of the total communication costs of the solutions found by KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.	34
3.2	Averages of the computational loads of the minimum and maximum loaded processors for the solutions found by KL-RB, KL-PM, SA, MFA heuristics, for randomly generated mapping problem instances.	35
3.3	Average execution times (in seconds) of KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.	36
4.1	Mean cut sizes of the solutions found by MFA, KL, and SA heuristics for randomly generated network partitioning problem instances.	56

1. INTRODUCTION

Some cognitive tasks as pattern recognition, associative recall, guiding of a mechanical hand are easily handled by biological neural networks whereas they remain as time consuming tasks for digital computers. This fact motivated scientists and opened a research area called Artificial Neural Networks (ANN). Scope of ANN includes understanding and modeling of biological neural networks, and designing artificial devices that have similar properties. Research on this area started with the early works of McCulloch and Pitts (1943), and has continued with varying levels of popularity until today. From the 1980s onwards, neural network models became the center of extensive study, and have seen an extraordinary growth of interest in their properties. Reasons for this increase in popularity are: better understanding gained on information processing in nature; increasing computer power which enables scientists to make better simulations and analysis of the models; growing interest in parallel computation and analog VLSI.

Research on ANN can be divided into two streams: first one deals with understanding and modeling of the biological neural networks, and second one exploits the information gained on biological neural networks for designing artificial devices or algorithms to perform tasks which are difficult for conventional computers. Until last few years, works on the second area were concentrated on learning and classification capability, and associative memory operation of the neural networks. Recent works by Hopfield and Tank [11, 12, 13, 31] show that solving NP-hard combinatorial optimization problems is another promising area for ANN. Hopfield and Tank proposed that, Hopfield type *continuous* and *deterministic* ANN model can be used for solving combinatorial optimization problems [11]. However, simulations of this model reveal the fact that it

is hard to obtain *feasible* solutions for large problem sizes. Many variants of the Hopfield Neural Network (HNN) have been designed [3, 30, 34] in order to improve the model for obtaining *feasible* and *good* solutions.

Combinatorial optimization problems constitute a large class, which is encountered in various disciplines. Optimization problems, in general, are characterized by searching for the *best* values of given variables to achieve a goal. In technical words, the objective is the minimization or maximization of a function, subject to some other constraint functions. A typical example is the general nonlinear programming problem, stated as: find $\mathbf{x} \in \mathbb{R}^n$ which

$$\begin{aligned} &\text{minimizes} && f(\mathbf{x}) \\ &\text{subject to} && g_i(\mathbf{x}) \geq 0 \quad i = 1, \dots, m \\ & && h_j(\mathbf{x}) = 0 \quad j = 1, \dots, p \end{aligned} \tag{1.1}$$

where f , g_i , h_j are general functions which map $\mathbb{R}^n \rightarrow \mathbb{R}$. The function f is called the cost function, and functions g_i and h_j are called constraint functions. Problems, for which the variables of the cost and constraint functions are *discrete*, are called combinatorial optimization problems. Some problems in this class can not be solved in polynomial time with the known methods. As the problem size increases, computing time needed to solve this kind of problems increases exponentially, resulting with intractable instances. This class of problems, called NP-hard optimization problems, are solved using heuristics. Heuristics are generally problem specific, computationally efficient algorithms. They do not guarantee to find optimal solution, but require much less computing time. The drawback of heuristics is that they usually get stuck in local minima.

In the last decade a powerful method, called Simulated Annealing (SA), has been developed for solving combinatorial optimization problems [18]. This method is the application of a successful statistical method, which is used to estimate the results of annealing process in statistical mechanics, to combinatorial optimization problems. SA is a general method (i.e. it is not problem specific) which guarantees to find the optimum solution if time is not limited. Time needed for simulated annealing is also too much and exact solutions of NP-hard problems still stay intractable. Nice property of simulated annealing

is that, it can be used as a heuristic to obtain near optimal solutions in limited time, and as the time limit is increased, quality of the obtained solutions also increase. SA has the capability of escaping from local minima if sufficient time is given. This method has been successfully applied to various NP-hard optimization problems [18, 20, 23].

The subject of this thesis is a recently proposed algorithm, called Mean Field Annealing (MFA) [22, 33, 34, 35]. MFA was originally proposed for solving the traveling salesperson problem [33, 34]. It combines the collective computation property of HNN with the annealing notion of SA. MFA is a general strategy and can be applied to various problems with suitable formulations. Work on MFA [4, 5, 21, 22, 34, 35] showed that, it can be successfully applied to combinatorial optimization problems. In this thesis, MFA is formulated for two well-known, NP-hard, combinatorial optimization problems: the mapping problem and the circuit partitioning problem.

The mapping problem arises while developing parallel programs for distributed-memory, message-passing parallel computers (multicomputers). In order to develop a parallel program for a multicomputer, first the problem is decomposed into a set of interacting sequential sub-problems (or tasks) that can be executed in parallel. Then, each one of these tasks is mapped to a processor of the parallel architecture, in such a way that the total execution time is minimized. This mapping phase is called the mapping problem [2], and is known to be NP-hard. In this thesis, MFA is formulated for solving the mapping problem, and its performance is compared with the performances of other well-known heuristics.

Partitioning of VLSI circuits are needed in various phases of VLSI design. Partitioning means to divide the components of a circuit into two or more evenly weighted partitions, such that the number of signal nets interconnecting them is minimized. This problem, called the circuit partitioning problem, is also an NP-hard combinatorial optimization problem. In this work, MFA is also formulated for solving the circuit partitioning problem, and the performance of the proposed algorithm is compared with the performances of other well-known heuristics.

Heuristics used for solving NP-hard combinatorial optimization problems as the mapping problem and the circuit partitioning problem are time consuming processes and parallelization of them is crucial. There is a large volume of research on the parallelization of such algorithms. One of the motivations in this work is to exploit the inherent parallelism in neural networks in order to obtain efficient parallel algorithms. MFA is a good candidate for efficient parallelization as it uses the collective computation property of HNN.

In order to develop a parallelization scheme, first the parallel computer that will be used must be classified. Classification of parallel architectures can be done according to their memory organization, the number of instruction streams supported, and the interconnection topology. Memory organization in parallel architectures can be divided into two main groups, shared-memory and distributed-memory architectures. In shared-memory architectures, which are named as multiprocessors, a common memory or a common address space is used by all processors. On the other hand, in distributed-memory architectures, processors can not access to a common memory space. Each processor has a local, isolated memory. Synchronization, coordination among processors and data exchange are achieved by message passing among processors. In this type of architectures, each processor may be viewed as an individual computer, hence they are called multicomputers.

Classification according to the interconnection topology determines how to handle communications among processors. Most commonly used topologies are mesh, hypercube and ring.

According to the number of instruction streams supported, parallel architectures can be divided into two groups. SIMD (Single Instruction stream Multiple Data stream) and MIMD (Multiple Instruction stream and Multiple Data stream) architectures. In a SIMD architecture, a central control processor broadcasts the instruction that will be executed to all processors. Each processor executes the same instruction using the data in its local memory. In MIMD architectures, each processor is able to fetch, decode and execute an instruction by itself, which can be different from the instructions executed by other processors.

In this work, MFA is parallelized for distributed-memory MIMD multicomputers, and implemented on a 3-dimensional iPSC/2 hypercube multicomputer. A d -dimensional hypercube consists of $P = 2^d$ processors with each processor being directly connected to d other (neighbor) processors [28]. The processors of the hypercube are labeled with d -bit binary numbers, and the binary label of each processor differs from that of its neighbor in exactly one bit. The parallelization schemes proposed in this work can also be used for SIMD multicomputers and other interconnection topologies with slight modifications.

In Chapter 2, HNN and SA are reviewed and a general formulation of MFA is given. Chapter 3 presents the proposed formulation of MFA for the mapping problem. Efficient implementation and parallelization of the proposed MFA algorithm is also addressed in this chapter. In Chapter 4, MFA is formulated for solving the circuit partitioning problem. Chapter 4 also presents efficient implementation and parallelization of the proposed algorithm. In Chapter 3 and 4, performances of the proposed MFA algorithms are evaluated in comparison with two well-known heuristics: simulated annealing and Kernighan-Lin. In Chapter 5, conclusions are stated.

2. THEORY

This chapter reviews previous works on Hopfield Neural Networks (HNN) and Simulated Annealing (SA) to give a better understanding of Mean Field Annealing (MFA). In Section 2.1 neural network models proposed by Hopfield are briefly discussed, and application of HNN to combinatorial optimization problems is described. A summary of the later works on HNN is also presented at the end of Section 2.1. Section 2.2 gives the general properties of simulated annealing and describes its application to combinatorial optimization problems. In Section 2.3, MFA algorithm is described, denoting the similarities with previously mentioned two methods.

2.1 Hopfield Neural Networks

One of the main reasons for the growing interest on neural networks in the last decade, is the Artificial Neural Network (ANN) model proposed by Hopfield [9]. Many ideas used in this model have precursors spread over the fifty years of research on neural networks. The importance of the work by Hopfield is that it brings them all together, using a physical analogy and a clear mathematical analysis, and gives a good view of the possible capabilities of the proposed model. Later, Hopfield proposed another model [10] that has the same properties of the original model, and looks very promising for VLSI implementations.

The original model [9] is a discrete, stochastic model, which uses two-state neurons with a stochastic updating algorithm. The continuous and deterministic model, which is proposed later [10], uses neurons with graded response, and

time evolution of the state of the system (change in the states of the neurons) is described by a differential equation. In these two models, an energy function, which always decreases as the system iterates, is defined. In his two consecutive papers [9, 10], Hopfield presented his ANN models as Content Addressable Memory (CAM) in order to explain their properties. In CAM model, minima of the energy function correspond to the stored words. Starting from a given initial state, the system is expected to reach one of these minima, which means to output one of the stored words in the CAM. CAM model of Hopfield can be regarded as an optimizing network: given an input, find one of the stored items which is the *closest* item to the given input. In his later works with Tank [11, 31] it is shown that well-known combinatorial optimization problems as the traveling salesperson problem, can also be solved by HNN.

2.1.1 Combinatorial Optimization Using Hopfield Neural Networks

Hopfield and Tank showed that, *continuous* and *deterministic* HNN has collective computational properties [11, 12, 13]. In collective computation, decisions taken to solve the problem is not determined by a single unit, but instead responsibility is distributed over a large number of simple, massively connected units. The nature of collective computation suggests that it might be particularly effective for problems that involve global interaction among different parts of the problem. NP-hard optimization problems are such problems. HNN can be used for solving a combinatorial optimization problem by choosing a representation scheme in which the output states of neurons can be decoded as a solution to the target problem. Then, HNN is constructed accordingly by choosing an energy function whose global minimum value corresponds to the *best solution* of the problem to be solved [11]. Hence, the constructed HNN is expected to compute the *best solution* to the target problem starting from a randomly chosen initial state by minimizing its energy function. General form of such an energy function (also called Hamiltonian of the system) is

$$H = cost + global\ constraint \quad (2.1)$$

Here, *cost* term represents the cost function of the optimization problem to be solved and *global constraint* term represents the constraint functions introduced to obtain feasible solutions. Exact solution of the problem corresponds to the global minimum of this energy function.

Motivation behind the works of Hopfield and Tank is to use hardware implementations of HNN to solve large optimization problems. It is a general method to simulate a model on computers before implementing it on hardware in order to observe and solve possible problems. In order to simulate HNN on a computer, first the equations of motion for the neural network are written from the state equations of the neurons. Then, these equations are solved for each neuron iteratively using a numerical method (usually Euler's method is used to compute the resulting differential equations). State of each neuron is computed in discrete time intervals until a stable state is found.

2.1.2 Problems of Hopfield Neural Networks

HNN have been applied to various optimization problems and reasonable results have been obtained for small size problems. However, simulations of this network reveals the fact that, it is hard to obtain *feasible solutions* for large problem sizes. Wilson and Pawley reports that, most of the simulation results give *infeasible tours* even for a 10-city traveling salesperson problem [36]. In fact, it is possible to obtain *feasible tours* by adjusting the parameters of the energy function (i.e., increasing the weights of the terms regarding feasibility), but, quality of the solutions deteriorate with such attempts. As is also indicated in [14], the problem of finding a balance among parameters of the energy function, in order to obtain *feasible* and *good* solutions, becomes harder as the problem size increases. Hence, the algorithm does not have a good scaling property, which is a very important performance criterion for heuristic optimization algorithms. Many attempts have been done to improve the performance of Hopfield neural network for obtaining *feasible* and *good* solutions. In one of them [3], number of terms in the energy function is decreased to increase the scalability of the algorithm. But also for that model, increase in the size of the problem causes the costs of the solutions to increase significantly.

Works by Szu [30] and Toomarian [32] are also modifications to HNN in which different energy functions are proposed. Recently, MFA is proposed as a successful alternative to HNN [22, 33, 34]. MFA algorithm combines the collective computation property of HNN and annealing notion of SA.

2.2 Simulated Annealing

SA is a powerful method which is used for solving hard optimization problems. In SA, an energy function that corresponds to the cost function of the problem to be solved is defined, similar to energy function defined for HNN. SA is a probabilistic hill-climbing method, which accepts uphill moves with a probability in order to escape from local minima. SA is derived using analogy to a successful statistical model of thermodynamic processes for growing crystals.

Configuration of a solid state material at a global energy minimum is a perfectly homogeneous crystal lattice. It is determined by experience that such configurations can be achieved using the process of *annealing* [20]. The solid-state material is heated to a high temperature until it reaches an amorphous liquid state. Then it is cooled slowly, according to a specific annealing schedule. If the initial temperature is sufficiently high to ensure a random state, and if the cooling schedule is sufficiently slow to guarantee that the equilibrium is reached at each temperature, final configuration of the material will be close to the perfect crystal with global energy minimum [20]. In thermodynamics, it is stated that, when thermal equilibrium at temperature T is reached, a state with energy E is attained with the Boltzmann probability

$$\frac{1}{Z(T)} e^{-\frac{E}{k_B T}} \quad (2.2)$$

where $Z(T)$ is a normalization factor and k_B is the Boltzmann constant [20].

There is a fine theoretical model which explains this physical phenomenon. During the annealing process the states of the atoms are perturbed by small random changes. If the change in state lowers the energy of the system, it is always accepted. If not, the change in configuration is accepted with a probability $e^{-\Delta E/k_B T}$. The probability of accepting perturbations causing increase

-
1. Get an initial configuration C
 2. Get initial temperature, and set $T = T_0$
 3. While not yet *frozen* DO
 - 3.1 While equilibrium at T is not yet reached DO
 - 3.1.1 Generate a random neighbor C' of C
 - 3.1.2 Let $\Delta E = E(C') - E(C)$
 - 3.1.3 If $\Delta E \leq 0$ (downhill move), set $C = C'$
 - 3.1.4 if $\Delta E > 0$ (uphill move), set $C = C'$ with
probability $e^{-\frac{\Delta E}{T}}$
 - 3.2 Update T according to the cooling schedule
-

Figure 2.1. Simulated annealing algorithm.

in energy decreases with the decreasing temperature, and minor modifications occur at lower temperatures. Experiments show that this model gives similar results as physical annealing process [20].

Kirkpatrick applied this model to optimization problems and called the resulting method SA. In transforming the physical model to computational model, energy function is replaced with the cost function of the optimization problem to be solved (note the similarity with HNN), and states of the matter are replaced with the legal configurations of the problem instance. Annealing schedule is controlled with a simulated temperature. Figure 2.1 illustrates the SA algorithm.

Although SA is a powerful method it has some problems. It requires a large amount of computing power because of the need for generating a large number of configurations, and very slow cooling in order to reach equilibrium at each temperature. Performance of the algorithm is closely related to the generation of neighboring configurations. It is an inherently sequential algorithm which

does not give good performance on parallel computers. It is hard to obtain good cooling schedules that results with good solutions in small amount of computer time.

2.3 Mean Field Annealing

MFA merges collective computation and annealing properties of the two methods described above, to obtain a general algorithm for solving combinatorial optimization problems. Mapping problems to MFA is identical to HNN. A neuron matrix is formed such that when neurons take their final values they represent a configuration in the solution space of the problem.

Mathematical analysis of MFA is done by analogy to Ising spin model, which is used to estimate the state of a system of particles or *spins* in thermal equilibrium. Spins in MFA algorithm are analogous to the neurons of HNN. This method was first proposed for solving the traveling-salesperson problem [33], and then it is applied to the graph partitioning problem [4, 5, 21, 35]. Here, general formulation of MFA algorithm [35] is given for the sake of completeness. In the Ising spin model, the energy of a system with S spins has the following form:

$$H(\mathbf{s}) = \frac{1}{2} \sum_{k=1}^S \sum_{l \neq k} \beta_{kl} s_k s_l + \sum_{k=1}^S h_k s_k \quad (2.3)$$

Here, β_{kl} indicates the level of interaction between spins k , l , and $s_k \in \{0, 1\}$ is the value of spin k . It is assumed that $\beta_{kl} = \beta_{lk}$ and $\beta_{kk} = 0$ for $1 \leq k, l \leq S$. At thermal equilibrium, spin average $\langle s_k \rangle$ of spin k can be calculated using Boltzmann distribution as follows

$$\langle s_k \rangle = \frac{1}{1 + e^{-\phi_k/T}} \quad (2.4)$$

Here, ϕ_k represents the *mean field* effecting on spin k , which can be computed using

$$\phi_k = -\frac{\partial \langle H(\mathbf{s}) \rangle}{\partial \langle s_k \rangle} \quad (2.5)$$

where the energy average $\langle H(\mathbf{s}) \rangle$ of the system is

$$\langle H(\mathbf{s}) \rangle = \sum_{k=1}^S \sum_{l \neq k} \beta_{kl} \langle s_k s_l \rangle + \sum_{k=1}^S h_k \langle s_k \rangle \quad (2.6)$$

-
1. Get initial temperature, and set $T = T_0$
 2. Initialize the spin averages $\langle \mathbf{s} \rangle = [\langle s_1 \rangle, \dots, \langle s_k \rangle, \dots, \langle s_S \rangle]$
 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 ϕ_k using

$$\phi_k = -\sum_{l \neq k} \beta_{kl} \langle s_l \rangle - h_k$$
 - 3.1.3 Update $\langle s_k \rangle$ using

$$\langle s_k \rangle = \{1 + e^{-\phi_k/T}\}^{-1}$$
 - 3.2 Update T according to the cooling schedule
-

Figure 2.2. Mean field annealing algorithm.

The complexity of computing ϕ_k using Eq. (2.5) and Eq. (2.6) is exponential [35]. However, for large number of spins, the *mean field approximation* can be used to compute the energy average as

$$\langle H(\mathbf{s}) \rangle = \frac{1}{2} \sum_{k=1}^S \sum_{l \neq k} \beta_{kl} \langle s_k \rangle \langle s_l \rangle + \sum_{k=1}^S h_k \langle s_k \rangle \quad (2.7)$$

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

$$\phi_k = -\frac{\partial \langle H(\mathbf{s}) \rangle}{\partial \langle s_i \rangle} = -\left(\sum_{l \neq k} \beta_{kl} \langle s_l \rangle + h_k\right) \quad (2.8)$$

Thus, the complexity of computing ϕ_k reduces to $O(S)$.

At each temperature, starting with initial spin averages, the mean field effecting on a randomly selected spin is found using Eq. (2.8). Then, spin average is updated using Eq. (2.4). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. The general form of the Mean Field Annealing algorithm derived from this iterative relaxation scheme is shown in Figure (2.2). MFA algorithm tries to

find equilibrium point of a system of S spins using annealing process similar to SA.

The state equations used in MFA are isomorphic to the state equations of the neurons in the HNN. A synchronous version of MFA, different from the algorithm given in Figure 2.2, can be derived by solving N difference equations for N spin values simultaneously. This technique is identical to the simulations of HNN done using numerical methods. Thus, evolution of a solution in a HNN is equivalent to the relaxation toward an equilibrium state affected by the MFA algorithm at a fixed temperature [35]. Hence MFA can be viewed as an annealed neural network derived from HNN.

HNN and SA methods have a major difference: SA is an algorithm implemented in software, whereas HNN is derived with a possible hardware implementation in mind. MFA is somewhere in between, it is an algorithm implemented in software, having potential for hardware realization [34, 35]. In this work, MFA is treated as a software algorithm as SA. Results obtained are comparable to other software algorithms, conforming this point of view.

3. MFA FOR THE MAPPING PROBLEM

In this chapter, Mean Field Annealing (MFA), is formulated for the mapping problem. In Section 3.1, the mapping problem is described and previous approaches used for solving the mapping problem are summarized. Section 3.2 presents a formal definition of the mapping problem by modeling the parallel program design process. Section 3.3 presents the proposed formulation of the MFA algorithm for the mapping problem. An efficient implementation scheme for the proposed algorithm is also described in Section 3.3.2. Section 3.4 presents the performance evaluation of the MFA algorithm for the mapping problem in comparison with two well-known mapping heuristics: simulated annealing and Kernighan-Lin. Finally, efficient parallelization of the MFA algorithm for the mapping problem is proposed in Section 3.5.

3.1 The Mapping Problem

Today, with the aid of VLSI technology, parallel computers not only exist in research laboratories, but are also available on the market as powerful, general purpose computers. Use of parallel computers in various applications, makes the problem of mapping parallel programs to parallel computers more crucial. The mapping problem arises while developing parallel programs for distributed-memory, message-passing parallel computers (multicomputers). In multicomputers, processors neither have shared memory nor have shared address space. Each processor can only access its local memory. Synchronization and coordination among processors are achieved through explicit message passing. Processors of a multicomputer are usually connected by utilizing one of

the well-known direct interconnection network topologies such as ring, mesh, hypercube, etc. These architectures have the nice scalability feature due to the lack of shared resources and the increasing bandwidth with increasing number of processors.

However, 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 and hence poor overall system performance. Processor idle time arises due to the uneven load balance in the distribution of the computational load among processors of the multicomputer. Parallel algorithm design for multicomputers can be divided into two phases: first phase is the decomposition of the problem into a set of interacting sequential sub-problems (or tasks) which can be executed in parallel. Second phase is mapping each one of these tasks to a processor of the parallel architecture in such a way that the total execution time is minimized. This mapping phase, named as the mapping problem [2], is very crucial in designing efficient parallel programs.

For a class of regular problems with regular interaction patterns, the mapping problem can be efficiently resolved by the judicious choice of the decomposition scheme. In such problems, chosen decomposition scheme yields an interaction topology that can be directly embedded to the interconnection network topology of the multicomputer. Such approaches can be referred as *intuitive* approaches. However, *intuitive* mapping approaches yield good results only for a restricted class of problems, under simplifying assumptions. The mapping problem is known to be NP-hard [15, 16]. Hence, heuristics giving sub-optimal solutions are used to solve the problem [1, 2, 6, 15, 16, 26]. Two distinct approaches have been considered in the context of mapping heuristics, one phase approaches and two phase approaches [6]. One phase approaches, referred to as *many-to-one mapping*, try to map tasks of the parallel program directly onto the processors of the multicomputer. In two phase approaches, *clustering* phase is followed by a *one-to-one mapping* phase. In the clustering phase, tasks of the parallel program is partitioned into as many equal weighted clusters as the number of processors of the multicomputer, while minimizing

the total weight of the inter-cluster interactions [26]. In the one-to-one mapping phase, each cluster is assigned to an individual processor of the multicomputer such that total inter-processor communication is minimized [26].

In two phase approaches, the problem solved in the clustering phase is identical to the multi-way graph partitioning problem. Graph partitioning is the balanced partitioning of the vertices of a graph into a number of bins, such that the total cost of the edges in the edge cut set is minimized. Kernighan-Lin (KL) heuristic [7, 17] is an efficient heuristic, originally proposed for the graph bipartitioning problem, which can also be used for clustering [6, 26]. KL heuristic is a non-greedy, iterative improvement technique that can escape from local minima by testing the gains of a sequence of moves in the search space before performing them. A variant of the KL heuristic can be used for solving one-to-one mapping problem encountered in the second phase [6].

Simulated Annealing (SA) can also be used as a one phase heuristic for solving many-to-one mapping problem [23, 29]. Successful applications of SA to the mapping problem is achieved in various works [23, 29]. It has been observed that the quality of the solutions obtained using SA are superior compared with the results of the other heuristics.

Heuristics proposed to solve the mapping problem are compute intensive algorithms. Solving the mapping problem can be thought as a preprocessing done before the execution of the parallel program on the parallel computer. If the mapping heuristic is executed sequentially, the execution time of this preprocessing can be included in the serial portion of the parallel program, which limits the efficiency that can be attained. In some cases, the sequential overhead caused by this preprocessing is not acceptable, and the need for the parallelization of the preprocessing arises. Efficient parallel mapping heuristics are needed in such cases. KL and SA heuristics are inherently sequential, hence hard to parallelize. Efficient parallelization of these algorithms remain as an important issue in parallel processing research.

In this chapter, Mean Field Annealing (MFA), is formulated for the many-to-one mapping problem. MFA has the inherent parallelism that exists in most of the neural network algorithms, which makes this algorithm a good candidate

for parallel mapping heuristics.

3.2 Modeling the Mapping Problem

Parallel program design phases are elaborated in this section in order to present a formal definition of the mapping problem. In the first phase of parallel algorithm design, problem is decomposed into a set of *atomic* tasks, such that the overall problem is modeled as a set of interacting tasks. Each atomic task is a sequential process to be executed by an individual processor of the parallel architecture. Selection of the decomposition scheme depends on the problem, algorithm to be used for the solution, and the architectural features of the target multicomputer.

In various classes of problems, interaction pattern among the tasks is static. Hence, the decomposition of the algorithm can be represented by a static task graph. Vertices of this graph represent the atomic tasks and the edge set represent the interaction pattern among the tasks. Relative computational costs of atomic tasks can be known or estimated priori to the execution of the parallel program. Hence, weights can be associated with the vertices to denote the computational costs of the corresponding tasks.

There are two different models used for modeling static inter-task communication patterns. These two models are referred as the Task Precedence Graph (TPG) model and Task Interaction Graph (TIG) model [16, 25]. TPG is a directed graph where directed edges represent execution dependencies. In this model, a pair of tasks connected by an edge can not be executed independently. Each edge denotes a pair of tasks: source task and destination task. The destination task can only be executed after the completion of the execution of the source task. Hence, in general, only the subsets of tasks which are unreachable from each other in the TPG can be executed independently.

In TIG, the set of interaction patterns are represented by undirected edges among vertices. In this model, each atomic task can be executed simultaneously and independently. Each edge denotes the need for the bidirectional interaction between corresponding pair of tasks at the completion of the execution of

these tasks. Edges may be associated with weights which denote the amount of bidirectional information exchange involved between pairs of tasks. TIG usually represents the repeated execution of the tasks with intervening inter-task interactions denoted by the edges.

The TIG model may seem to be unrealistic for general applications since it does not consider the temporal interaction dependencies among the tasks [25]. However, there are various classes of problems which can be successfully modeled with the TIG model. For example, iterative solution of systems of equations, and problems arising in image processing and computer graphics applications can be represented by TIG. In this work, mapping of problems which can be represented by TIG model is addressed.

Second phase of the parallel algorithm design is the assignment of the individual tasks to the processors of the parallel architecture, so that the execution time of the parallel program is minimized. This problem is referred as the mapping problem. In order to solve the mapping problem, 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. In fact, POG is a graphical representation of the interconnection topology utilized for the organization of the processors of the parallel architecture. In general, 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.

In a multicomputer architecture, each adjacent pair of processors communicate with each other over the communication link connecting them. Such communications are referred as *single-hop* communications. However, each non-adjacent pair of processors can also communicate with each other via *software* or *hardware routing*. Such communications are referred as *multi-hop* communications. Multi-hop communications are usually routed in a static manner over the shortest path of links between the communicating pairs of processors. Communications between non-adjacent pairs of processors can be associated with relative unit communication costs. Unit communication cost is defined

as the communication cost per unit of information. Unit communication cost between a pair of processors will be a function of the shortest path between these processors and the routing scheme used for multi-hop communications. For example, intermediate processors in the communication path are interrupted in software routing so that each multi-hop communication is realized as a sequence of single-hop messages. Hence, in software routing, the unit communication cost is linearly proportional to the shortest path distance between the pair of communicating processors. Note that, in this communication model, unit communication costs between adjacent pairs of processors are taken to be unity.

Hence, 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. As is mentioned earlier, PCG can easily be constructed using the topological properties of the POG and the *routing* scheme utilized for inter-processor communication. In the PCG, edges between pairs of nodes representing the adjacent pairs of processors denote physical links whereas edges between pairs of nodes representing non-adjacent pairs of processors denote virtual communication links (i.e. communication paths) established for routing multi-hop communications.

The objective in mapping TIG to PCG is the minimization of the expected execution time of the parallel program on the target architecture represented by the TIG and the PCG respectively. Thus, the mapping problem can be modeled as an optimization problem by associating the following quality measures with a good mapping :

- Interprocessor communication overhead should be minimized. Tasks which have high interaction, i.e., large amount of data exchange, should be in the same processor or nearby processors.
- Computational load should be uniformly distributed among processors. Computational load assigned to each processor should be equal as much as possible in order to minimize processor idle time.

The parallel execution time is expected to decrease as these criteria are satisfied.

A mapping problem instance can be formally defined as follows. An instance of the mapping problem includes two undirected graphs, Task Interaction Graph (TIG) and Processor Communication Graph (PCG). The TIG $G_T(V, E)$, has $|V| = N$ vertices labeled as $(1, 2, \dots, i, j, \dots, N)$. Vertices of the TIG represent the atomic tasks of the parallel program. Vertex weight w_i denotes the computational cost associated with task i for $1 \leq i \leq N$. Edge weight e_{ij} denotes the volume of interaction between tasks i and j connected by edge $(i, j) \in E$. The PCG $G_P(P, D)$, is a complete graph with $|P| = K$ nodes and $|D| = \binom{K}{2}$ edges. Nodes of the PCG, labeled as $(1, 2, \dots, p, q, \dots, K)$, represent the processors of the target multicomputer. Edge weights d_{pq} , for $1 \leq p, q \leq K$ and $p \neq q$, denote the unit communication cost between processors p and q .

Given an instance of the mapping problem with TIG, $G_T(V, E)$, and PCG, $G_P(P, D)$, question is to find a many-to-one mapping function $M : V \rightarrow P$, which assigns each vertex of the graph G_T to a unique node of graph G_P , and minimizes the total interprocessor communication cost (CC)

$$CC = \sum_{(i,j) \in E} e_{ij} d_{M(i)M(j)} \quad (3.1)$$

while having the computational load (CL_p : computational load of processors p)

$$CL_p = \sum_{i \in V, M(i)=p} w_i, \quad 1 \leq p \leq K \quad (3.2)$$

of each processor balanced. Here, $M(i) = p$ denotes the label (p) of the processor that task i is mapped to. In Eq. (3.1), each edge (i, j) of the TIG contributes to the communication cost (CC), only if vertices i and j are mapped to two different nodes of the PCG, i.e., $M(i) \neq M(j)$. The amount of contribution is equal to the product of the volume of interaction e_{ij} between these two tasks and the unit communication cost d_{pq} between processors p and q where $p = M(i)$ and $q = M(j)$. The computational load of a processor is the summation of the weights of the tasks assigned to that processor. Perfect load balance is achieved if $CL_p = (\sum_{i=1}^N w_i)/K$ for $1 \leq p \leq K$. Balancing of the

computational loads of the processors can be explicitly included in the cost function using a term which is minimized when the loads of the processors are equal. Another scheme is to include balancing criteria implicitly in the algorithm. Figure 3.1 illustrates a sample mapping problem instance with $N = 8$ tasks to be mapped onto $K = 4$ processors. Figure 3.1(a) shows the TIG with $N = 8$ tasks. Figure 3.1(b) shows the POG for a 2-dimensional hypercube, and Figure 3.1(c) shows the corresponding PCG. In Figure 3.1, numbers inside the circles denote the vertex labels, and numbers within the parenthesis denote the vertex or edge weights. Binary labeling of the 2-dimensional hypercube is also given in Figure 3.1(b). Note that unit communication cost assignment to edges is performed assuming software routing protocol for multi-hop communications. A solution to the mapping problem instance shown in Figure 3.1 is

i	1	2	3	4	5	6	7	8
$M(i)$	1	1	4	3	2	4	2	3

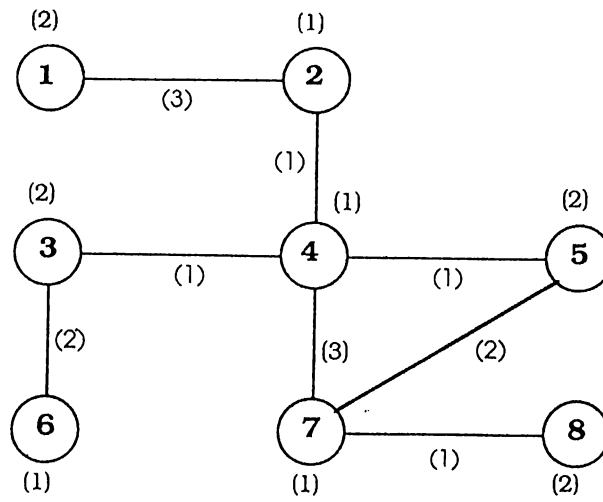
Communication cost of this solution can be calculated as

$$CC = \sum_{(i,j) \in E} e_{ij} d_{M(i)M(j)} = 8 \quad (3.3)$$

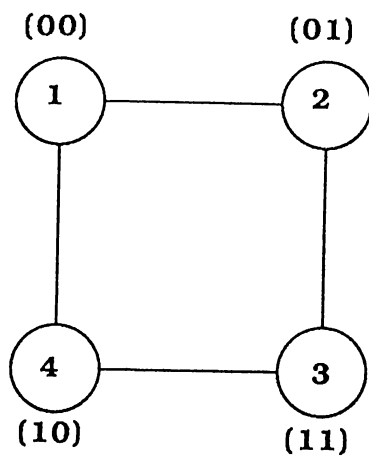
Computation loads of the processors are $CL_p = 3$ for $1 \leq p \leq 4$. Hence, perfect load balance is achieved since, $(\sum_{i=1}^8 w_i)/4 = 3$.

3.3 Solving the Mapping Problem Using MFA

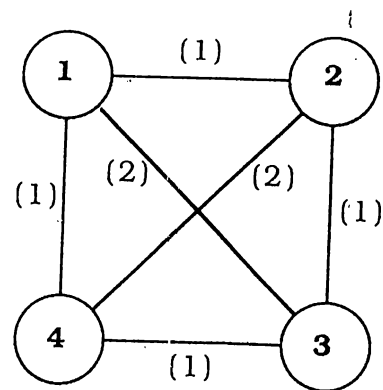
In this section, a formulation of the Mean Field Annealing (MFA) algorithm for the mapping problem is proposed. The TIG and PCG models described in Section 3.2 are used to represent the mapping problem. The formulation is first presented for problems modeled by dense TIGs. The modification in the formulation for mapping problems that can be represented by sparse TIGs is presented later. In this section, an efficient implementation scheme for the proposed formulation is also presented.



(a)



(b)



(c)

Figure 3.1. A mapping problem instance, with (a) TIG, (b) POG (which represents a 2-dimensional hypercube) and (c) PCG.

3.3.1 Formulation

A spin matrix, which consists of N task-rows and K processor-columns, is used as the representation scheme. Hence, $N \times K$ 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 . Here, s_{ip} is a continuous variable in the range $0 \leq s_{ip} \leq 1$. When MFA algorithm reaches to a solution, spin values converge to 1 or 0 indicating the result. If s_{ip} is 1, this means that task i is mapped to processor p . For example, a solution to the mapping instance given in Figure 3.1 can be represented by the following $N \times K$ spin matrix.

$$\begin{array}{c}
 \text{K Processors} \\
 \hline
 \begin{array}{cccc}
 1 & 2 & 3 & 4
 \end{array} \\
 \\
 \begin{array}{c}
 \text{N Tasks}
 \end{array}
 \left\{
 \begin{array}{ccccc}
 1 & 1 & 0 & 0 & 0 \\
 2 & 1 & 0 & 0 & 0 \\
 3 & 0 & 0 & 0 & 1 \\
 4 & 0 & 0 & 1 & 0 \\
 5 & 0 & 1 & 0 & 0 \\
 6 & 0 & 0 & 0 & 1 \\
 7 & 0 & 1 & 0 & 0 \\
 8 & 0 & 0 & 1 & 0
 \end{array}
 \right.
 \end{array}$$

Note that, this solution is identical to the solution given at the end of Section 3.2.

Following energy (i.e., cost) function is proposed for the mapping problem

$$H(\mathbf{s}) = \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N \sum_{p=1}^K \sum_{q \neq p}^K e_{ij} s_{ip} s_{jq} d_{pq} + \frac{r}{2} \sum_{i=1}^N \sum_{j \neq i}^N \sum_{p=1}^K s_{ip} s_{jp} w_i w_j \quad (3.4)$$

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. Weight of the edge between processors p and q in the PCG is represented by d_{pq} .

Under the mean field approximation, the expression $\langle H(\mathbf{s}) \rangle$ for the expected value of objective function given in Eq. (3.4) will be similar to the expression given for $H(\mathbf{s})$ in Eq. (3.4). However, in this case, s_{ip} , s_{iq} and s_{jp} should be replaced with $\langle s_{ip} \rangle$, $\langle s_{iq} \rangle$ and $\langle s_{jp} \rangle$ respectively. For the sake of simplicity, s_{ip}

is used to denote the expected value of spin (i, p) (i.e., spin average $\langle s_{ip} \rangle$) in the following discussions.

In Eq. (3.4), 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, under the mean field approximation. 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 the tasks i and j to different processors. Note that, in Eq. (3.4), the first quadruple summation term covers all processor pairs in the PCG for each edge pair in the TIG. Hence, the first quadruple summation term denotes the total interprocessor communication cost for a mapping represented by an instance of the spin matrix. Then, minimization of the first quadruple summation term corresponds to the minimization of the interprocessor communication overhead for the given mapping problem instance.

Second triple summation term in Eq. (3.4) computes the summation of the inner products of the weights of the tasks mapped to individual processors for a mapping. Global minimum of the second triple summation term occurs when equal amount of task weights are mapped to each processor. If there is an imbalance in the mapping, second triple summation term increases with the square of the weight of the imbalance, penalizing imbalanced mappings. The parameter r in Eq. (3.4) is introduced to maintain a balance between the two optimization objectives of the mapping problem.

Using the mean field approximation described in Eq. (2.8), the expression for the mean field ϕ_{ip} experienced by spin (i, p) can be found to be

$$\phi_{ip} = - \sum_{j \neq i} \sum_{q \neq p}^N e_{ij} s_{jq} d_{pq} - r \sum_{j \neq i}^N s_{jp} w_i w_j \quad (3.5)$$

In a feasible mapping, each task should be mapped exclusively to a single processor. However, there exists no penalty term in Eq. (3.4) to handle this feasibility constraint. This feasibility constraint is explicitly handled while updating the spin values. Note that, from Eq. (2.4), individual spin average s_{ip} is proportional to $e^{\phi_{ip}/T}$, i.e. $s_{ip} \propto e^{\phi_{ip}/T}$. Then, s_{ip} is normalized as

$$s_{ip} = \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^K e^{\phi_{iq}/T}} \quad (3.6)$$

This normalization enforces the summation of each row of the spin matrix to be equal to unity. Hence, it is guaranteed that all rows of the spin matrix will have only one spin with output value 1 when the system is stabilized.

Eq. (3.5) can be interpreted in the context of the mapping problem as follows. First double summation represents the rate of increase expected in the total interprocessor communication cost by mapping task i to processor p . Second summation represents the rate of increase in the computational load balance cost associated with processors p by mapping task i to processor p . Hence, $-\phi_{ip}$ may be interpreted as the expected rate of decrease in the overall quality of the mapping by mapping task i to processor p . Then, in Eq. (3.6), s_{ip} is updated such that the probability of task i being mapped to processor p increases with increasing mean field ϕ_{ip} experienced by spin (i, p) . Hence, the MFA heuristic can be considered as a gradient-descent type algorithm in this context. However, it is also a stochastic algorithm similar to SA due to the random spin update scheme and the annealing process.

In the general MFA algorithm given in Figure 2.2, a randomly chosen spin is updated at a time. However, in the proposed formulation of the MFA for the mapping problem, K spins of a randomly chosen row of the spin matrix are updated at a time. This update operation is performed as follows. Mean fields ϕ_{ip} , ($1 \leq p \leq K$) experienced by the spins at the i -th row of the spin matrix are computed by using Eq. (3.5) for $p = 1, 2, \dots, K$. Then, the spin averages s_{ip} , $1 \leq p \leq K$ are updated using Eq. (3.6) for $p = 1, 2, \dots, K$. 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 [34]. If energy function H is not decreasing after a certain number of consecutive spin-row updates, this means that the system is stabilized for that temperature [34]. Then, T is decreased according to the cooling schedule, and iteration process is re-initiated. Note that, the computation of the energy difference ΔH , necessitates the computation of H (Eq. (3.4)) at each iteration. The complexity of computing H is $O(N^2 \times K^2)$, which drastically increases the complexity of one iteration of MFA. Here, we propose an efficient scheme which reduces the

complexity of energy difference computation by an asymptotical factor.

The *incremental* energy change δH_{ip} because of the *incremental* change δs_{ip} in the value of an individual spin (i, p) is

$$\delta H = \delta H_{ip} = \phi_{ip} \delta s_{ip} \quad (3.7)$$

due to Eq. (2.5). Since, $H(\mathbf{s})$ is linear in s_{ip} (see Eq. (3.4)), above equation is valid for any amount of change Δs_{ip} in the value of spin (i, p) , that is

$$\Delta H = \Delta H_{ip} = \phi_{ip} \Delta s_{ip} \quad (3.8)$$

At each iteration of the MFA algorithm, K spin values are updated in a synchronous manner. Hence, Eq. (3.8) is valid for all spin updates performed in a particular iteration (i.e. for $1 \leq p \leq K$). Thus, energy difference due to the spin-row update operation in a particular iteration can be computed as

$$\Delta H = \Delta H_i = \sum_{p=1}^K \phi_{ip} \Delta s_{ip} \quad (3.9)$$

where $\Delta s_{ip} = s_{ip}^{new} - s_{ip}^{old}$. The complexity of computing Eq. (3.9) is only $O(K)$ since mean field (ϕ_{ip}) values are already computed for the spin updates.

The formulation of the MFA algorithm for the mapping problem instances with sparse TIGs is done as follows. The expression given for ϕ_{ip} (Eq. (3.5)) can be modified for sparse TIGs as

$$\phi_{ip} = - \sum_{j \in Adj(i)} \sum_{q \neq p}^K e_{i,j} s_{jq} d_{pq} - r \sum_{j \neq i}^N s_{jp} w_i w_j \quad (3.10)$$

Here, $Adj(i)$ denotes the set of tasks connected to task i in the given TIG. Note that, sparsity of the TIG can only be exploited in mean field computations since spin update operations given in Eq. (3.6) are dense operations which are not effected by the sparsity of the TIG.

The steps of the MFA algorithm for solving the mapping problem is given in Figure 3.2. Complexity of computing first double summation terms in Eq. (3.5) and Eq. (3.10) are $O(N \times K)$ and $O(d_{avg} \times K)$ for dense and sparse TIGs respectively. Here, d_{avg} denotes the average degree of the vertices of the sparse TIG. Second summation operations in Eq. (3.5) and Eq. (3.10) are both $O(N)$ for dense and sparse TIGs. Then, complexity of a single mean field computation

-
1. Get initial temperature, and set $T = T_0$
 2. Initialize the spin averages $\mathbf{s} = [s_{11}, \dots, s_{ip}, \dots, s_{NK}]$
 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

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

$$s_{ip}^{new} = e^{\phi_{ip}/T} / \sum_{p=1}^K e^{\phi_{ip}/T} \quad \text{for } 1 \leq p \leq K$$
 - 3.1.5 Compute the change in energy 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

$$s_{ip} = s_{ip}^{new} \quad \text{for } 1 \leq p \leq K$$
 - 3.2 $T = \alpha \times T$
-

Figure 3.2. MFA algorithm for the mapping problem.

is $O(N \times K)$ and $O(d_{avg} \times K + N)$ for dense (Eq. (3.5)) and sparse (Eq. (3.10)) TIGs respectively. Hence, complexity of mean field computations for a spin row is $O(N \times K^2)$ for dense TIGs, and $O(d_{avg} \times K^2 + N \times K)$ for sparse TIGs (step 3.1.2 in Figure 3.2). Spin update computations (steps 3.1.3, 3.1.4 and 3.1.6) and energy difference computation (step 3.1.5) are both $O(K)$ operations. Hence, the overall complexity of a single MFA iteration is $O(N \times K^2)$ for dense TIGs, and $O(d_{avg} \times K^2 + N \times K)$ for sparse TIGs.

3.3.2 An Efficient Implementation Scheme

As is mentioned earlier, the MFA algorithm proposed for the mapping problem is an iterative process. The complexity of a single MFA iteration is mainly due to the mean field computations. In this section, we propose an efficient implementation scheme which reduces the complexity of the mean field computations and hence the complexity of the MFA iteration by asymptotical factors.

Assume that, i -th spin-row is selected at random for update in a particular iteration. The expression given for ϕ_{ip} (Eq. (3.5)) can be rewritten by changing the order of summations of the first double summation term as

$$\begin{aligned}\phi_{ip} &= -\sum_{q \neq p}^K d_{pq} \sum_{j \neq i}^N e_{i,j} s_{jq} - r \sum_{j \neq i}^N s_{jp} w_i w_j \\ &= -\sum_{q \neq p}^K d_{pq} \lambda_{iq} - r \psi_{ip}\end{aligned}\quad (3.11)$$

where

$$\lambda_{iq} = \sum_{j \neq i}^N e_{i,j} s_{jq} \quad (3.12)$$

$$\psi_{ip} = \sum_{j \neq i}^N s_{jp} w_i w_j \quad (3.13)$$

Here, λ_{iq} represents the rate of increase expected in the interprocessor communication by mapping task i to a processor other than q (for the current mapping on processor q), assuming uniform unit communication cost between all pairs of processors in PCG. Similarly, ψ_{ip} represents the rate of increase expected in the computational load balance cost associated with processor p , by mapping task i to processors p (for the current mapping on processor p).

For an efficient implementation, the overall mean field computation involved in a single iteration can be computed using the following matrix equation

$$\Phi_i = -D \times \Lambda_i - r\Psi_i \quad (3.14)$$

$$= -\Theta_i - r\Psi_i \quad (3.15)$$

Here, D is a $K \times K$ adjacency matrix representing PCG (i.e. $D_{pq} = d_{pq}$), and Φ_i , Λ_i , Ψ_i and Θ_i are column vectors with K elements, where

$$\begin{aligned} \Phi_i &= [\phi_{i1}, \dots, \phi_{ip}, \dots, \phi_{iK}]^T & \Lambda_i &= [\lambda_{i1}, \dots, \lambda_{ip}, \dots, \lambda_{iK}]^T \\ \Psi_i &= [\psi_{i1}, \dots, \psi_{ip}, \dots, \psi_{iK}]^T & \Theta_i &= [\theta_{i1}, \dots, \theta_{ip}, \dots, \theta_{iK}]^T \end{aligned} \quad (3.16)$$

The complexity analysis of the proposed implementation scheme for dense TIGs is as follows. Complexity of computing λ_{ip} and ψ_{ip} are both $O(N)$. Complexity of constructing Λ_i and Ψ_i vectors are both $O(N \times K)$, since both vectors contain K such entries. Complexity of computing the matrix-vector product required in Eq. (3.14) is $O(K^2)$. Hence, the overall complexity of computing the Φ_i vector (Eq. (3.14)) reduces to $O(N \times K + K^2) = O(N \times K)$, since $N \gg K$ in general. The complexity of K spin updates and the computation of ΔH are both $O(K)$. Thus, the proposed scheme reduces the computational complexity of a single MFA iteration to $O(N \times K)$ for dense TIGs with $N \gg K$.

The complexity analysis of the proposed implementation for sparse TIGs is as follows. Note that, the sparsity of the TIG can only be exploited in the computation of λ_{iq} 's since

$$\lambda_{iq} = \sum_{j \in Adj(i)}^N e_{ij} s_{jq} \quad (3.17)$$

for sparse TIGs. Hence, the complexity of computing an individual λ_{iq} is only $O(d_{avg})$. Thus, the complexity of constructing the Λ_i vector reduces to $O(d_{avg} \times K)$. The complexity of computing the Θ_i vector in Eq. (3.15) reduces to $O(d_{avg} \times K + K^2)$. However, the complexity of constructing the Ψ_i vector required in Eq. (3.15) is $O(N \times K)$, dominating the overall complexity of the mean field computations. The complexity of computing the Ψ_i vector can be reduced as follows. The computation of ψ_{ip} in Eq. (3.13) can be re-formulated

as

$$\begin{aligned}\psi_{ip} &= \sum_{j \neq i}^N s_{jp} w_i w_j = w_i \sum_{j \neq i}^N w_j s_{jp} = w_i \left(\sum_{j=1}^N w_j s_{jp} - w_i s_{ip} \right) \\ \psi_{ip} &= w_i \gamma_p - w_i^2 s_{ip}\end{aligned}\tag{3.18}$$

where

$$\gamma_p = \sum_{j=1}^N w_j s_{jp}\tag{3.19}$$

Here, γ_p represents the computational load of processor p , for the current mapping on processor p . Note that, computationally, γ_p represents weighted sum of spin values of the p -th column of the spin matrix. Hence, initial γ_p value of each column p ($1 \leq p \leq K$) can be computed by using Eq. (3.19) for the initial spin values. Then, γ_p values can be updated at the end of each iteration (i.e. after spin updates) by using

$$\gamma_p^{new} = \gamma_p^{old} - w_i s_{ip}^{old} + w_i s_{ip}^{new}\tag{3.20}$$

for $1 \leq p \leq K$.

The computation of initial γ_p values can be excluded from the complexity analysis since they are computed only once at the very beginning of the algorithm. In this scheme, the computation of an individual ψ_{ip} using Eq. (3.18) is an $O(1)$ operation. Hence, the construction of the Ψ_i vector required in Eq. (3.14) becomes an $O(K)$ operation. Thus, the complexity of computing the mean field values reduces to $O(d_{avg} \times K + K^2)$. Note that, the update of an individual γ_p value (using Eq. (3.20)) at the end of the iteration is an $O(1)$ operation. Hence, the overall complexity of γ_p updates is $O(K)$ since K weighted column sums should be updated at each iteration. Note that, complexity of spin updates and energy difference computation are also $O(K)$ for sparse TIGs. Hence, the implementation scheme proposed for sparse TIGs reduces the complexity of a single MFA iteration to $O(d_{avg} \times K + K^2)$.

3.4 Performance of Mean Field Annealing Algorithm

This section presents the performance evaluation of the Mean Field Annealing (MFA) algorithm for the mapping problem, in comparison with two well-known

mapping heuristics: Simulated Annealing (SA) and Kernighan-Lin (KL). Each algorithm is tested using randomly generated mapping problem instances. In the following sections implementations are described in order to give a better understanding of the discussed algorithms.

3.4.1 MFA Implementation

MFA algorithm described in the previous section (Figure 3.2) is implemented for testing the performance of the algorithm. Cooling process is started from an initial temperature which is found experimentally. For the mapping problem instances used in the experiments, initial temperature T_0 is found to be varying between $1 \leq T_0 \leq 10$. Coefficient r which determines the balance between two optimization criteria is also found experimentally, varying between $0.1 \leq r \leq 1.5$. At each temperature, iterations continued until $\Delta H < \epsilon$ for L consecutive iterations. L is set equal to N initially. Parameter ϵ is chosen to be between $10^{-3} \leq \epsilon \leq 10^{-1}$. Temperature is decreased using $\alpha = 0.9$ until T is less than $T_0/1.5$. Then, L is set to $L/4$ and α is set to 0.5 and cooling is continued until T is less than $T_0/5.0$. Resulting spin values after this cooling operation are set to 0 if they are less than 0.5 and set to 1 if they are greater than 0.5. Then the result is decoded as described in Section 3.3 and the resulting mapping is found.

3.4.2 Kernighan-Lin Implementation

Kernighan-Lin heuristic is not directly applicable to the mapping problem since it was originally proposed for graph bipartitioning. In order to apply KL heuristic to the mapping problem a two phase approach is used. In the first phase, task interaction graph $G_T(V, E)$ is partitioned to K clusters, where K is equal to the number of processors. These K clusters are then mapped to processor graph $G_P(P, D)$ using a one-to-one mapping heuristic in the second phase. One-to-one mapping heuristic used in this work is a variant of KL heuristic.

For the clustering phase, Kernighan-Lin heuristic is implemented efficiently

as described by Fiduccia and Mattheyses [7]. In order to apply KL to K -way graph partitioning two schemes are used. First one, partitioning by recursive bisection (KL-RB), recursively partitions the initial graph to two partitions until K partitions are obtained. Other scheme, partitioning by pairwise min-cut (KL-PM), starts with an initial K -way partitioning and then minimizes the cutsizes between each pair of partitions until no improvement can be done. In KL heuristic balancing of the work load of processors is done implicitly by the algorithm. When moving one node from one partition to another, weights of the partitions are tested and moves causing intolerable imbalance are rejected.

In the beginning of second phase, K clusters formed in the first phase are mapped to the K processors of the multicomputer randomly. After this initial mapping, communication cost is minimized by performing a sequence of cluster swaps. An individual cluster swap corresponds to interchanging the mapping of a pair of clusters.

3.4.3 Simulated Annealing Implementation

Simulated Annealing algorithm, implemented for solving the mapping problem, uses the one phase approach to map the TIG onto PCG. In simulated annealing, starting from a randomly chosen initial configuration, configuration space is searched for the best solution using a probabilistic hill climbing algorithm. A configuration of the mapping problem is a mapping between TIG and PCG, which assigns each task in TIG to a processor in PCG. In order the search the configuration space, neighborhood of a configuration must be defined. For the implementation in this work, neighborhood of a configuration consists of all configurations which results with moving one vertex (task) of the TIG from the maximum loaded node (processor) of the PCG to another node of PCG. At each iteration of the simulated annealing algorithm, one of the possible moves is chosen randomly as a candidate move. Then the resulting decrease in the total communication cost after performing the candidate move is calculated without changing the configuration. If the candidate move decreases the cutsizes, it is realized. If candidate move increases the cutsizes, then it is realized with a probability which decreases with the increasing positive difference caused in the

total cutsize by that move. Acceptance probability of the moves that increases the cost is controlled with a temperature parameter T which is decreased using an annealing schedule. Hence, as the annealing proceeds acceptance probability of uphill moves decreases. Cooling schedule used in the implementation of SA algorithm is similar to the schedule given in [22].

3.4.4 Experimental Results

In this section, performance of the MFA algorithm is discussed in comparison with SA and KL algorithms. These heuristics are experimented for mapping randomly generated TIGs onto mesh and hypercube connected multicomputers.

Six test TIGs are generated with $N = 200$ and 400 vertices. Vertices of these TIGs are weighted by assigning a randomly chosen integer weight between 1 and 10 to each vertex ($1 \leq w_i \leq 10$, for $1 \leq i \leq N$). Interaction patterns among the vertices of these TIGs are constructed as follows. A maximum vertex degree, d_{max} , is selected for each test TIG ($d_{max} = 8, 16, 32$) such that, degree d_i of each vertex i is a randomly chosen value between 1 and d_{max} (i.e. $1 \leq d_i \leq d_{max}$, for $1 \leq i \leq N$). Then, each vertex i of TIG is connected to d_i randomly chosen vertices. Resulting edges are weighted randomly with integer values varying between 1 and 10. These TIGs are mapped to 3-, 4-, 5-dimensional hypercubes and 4×4 , 4×8 two dimensional mesh multicomputers. PCGs corresponding to these interconnection topologies are constructed assuming software routing as is described in Section 3.2.

Tables 3.1, 3.2 and 3.3 illustrate the performance results of KL-RB, KL-PM, SA and MFA heuristics for the generated mapping problem instances. In these tables, N and $|E|$ denote the number of vertices and edges in the test TIGs respectively, and K denotes the number of processors on the target PCG. Interconnection topology of the target PCG is denoted by T , where H denotes the hypercube interconnection topology and M denotes the mesh interconnection topology. Each algorithm is executed 10 times for each problem instance, starting from different, randomly chosen initial configurations. Averages of the results are illustrated in Tables 3.1, 3.2 and 3.3.

Table 3.1. Averages of the total communication costs of the solutions found by KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE COMMUNICATION COST			
N	$ E $	K	T	KL-RB	KL-PM	SA	MFA
200	544	8	H	1807.4	1846.0	1595.1	1671.4
200	544	16	H	2819.9	2747.1	2180.0	2333.4
200	544	32	H	4098.8	4710.4	2879.0	3181.6
200	1120	8	H	5421.9	5494.7	4947.8	5092.4
200	1120	16	H	7742.4	7816.1	6699.1	6840.3
200	1120	32	H	10377.1	11280.2	8495.7	9200.3
200	2152	8	H	12721.6	12959.0	12018.5	11956.2
200	2152	16	H	17828.9	17859.9	16201.2	16261.2
200	2152	32	H	23127.6	24260.3	20407.0	20586.0
400	1227	8	H	4360.6	4444.5	3772.3	4235.6
400	1227	16	H	6096.0	6073.2	5086.4	5615.9
400	1227	32	H	8420.2	7999.9	6485.0	7184.0
400	2283	8	H	11247.1	11491.5	10152.1	10744.3
400	2283	16	H	15566.7	15896.9	13626.7	14197.5
400	2283	32	H	20543.8	20527.1	17169.8	18209.6
400	4298	8	H	25318.3	25832.1	23507.6	23561.1
400	4298	16	H	34590.6	35395.4	31427.2	32127.6
400	4298	32	H	45053.8	45098.1	39453.0	40133.8
200	544	16	M	3364.2	3318.7	2659.7	2996.0
200	544	32	M	5618.7	6822.5	4260.4	4580.0
200	1120	16	M	9234.2	9318.2	8432.3	8121.7
200	1120	32	M	14659.9	16476.4	13556.0	12456.9
400	1227	16	M	7341.4	7357.0	6293.0	6745.0
400	1227	32	M	12207.4	11758.6	9924.8	10780.0
400	2283	16	M	18670.9	19133.0	17480.1	16631.6
400	2283	32	M	29827.0	30156.3	28319.1	26078.2

Table 3.2. Averages of the computational loads of the minimum and maximum loaded processors for the solutions found by KL-RB, KL-PM, SA, MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE MIN-MAX LOAD							
N	$ E $	K	T	KL-RB		KL-PM		SA		MFA	
200	544	8	H	125.0	153.3	126.8	150.2	135.1	142.7	132.2	143.6
200	544	16	H	59.0	80.0	63.4	75.0	64.0	74.4	54.9	83.1
200	544	32	H	28.6	41.6	30.8	37.0	29.2	41.0	28.4	41.6
200	1120	8	H	121.4	155.6	125.7	150.6	134.1	142.9	127.0	149.4
200	1120	16	H	59.1	81.3	63.3	74.9	64.0	74.9	61.6	77.8
200	1120	32	H	28.6	42.4	29.4	37.0	28.2	42.8	30.7	39.4
200	2152	8	H	120.2	156.9	124.4	149.8	133.3	143.5	128.9	149.2
200	2152	16	H	57.4	81.8	62.0	74.0	63.1	67.9	60.7	79.4
200	2152	32	H	27.3	42.8	31.0	37.0	27.8	40.4	25.8	44.1
400	1227	8	H	250.9	319.4	259.2	313.0	281.7	290.6	281.6	289.9
400	1227	16	H	124.3	164.6	129.4	156.8	138.1	148.8	135.6	150.4
400	1227	32	H	60.2	87.0	64.6	78.0	66.0	77.0	58.7	86.7
400	2283	8	H	241.7	313.0	248.4	300.6	280.1	270.7	266.9	284.4
400	2283	16	H	115.7	159.8	124.3	149.9	132.6	143.2	126.5	149.3
400	2283	32	H	56.4	84.5	62.2	74.0	63.5	74.0	62.4	76.4
400	4298	8	H	253.6	331.0	261.6	318.8	285.4	298.3	273.4	309.7
400	4298	16	H	122.2	169.9	131.2	158.5	138.8	153.0	135.3	155.2
400	4298	32	H	59.5	88.9	65.0	79.0	67.3	77.7	58.2	87.6
200	544	16	M	58.6	79.7	63.2	74.8	63.2	74.4	62.8	76.4
200	544	32	M	28.7	41.4	31.0	37.0	29.1	39.5	26.0	42.6
200	1120	16	M	58.5	81.0	63.2	75.0	64.0	75.8	61.3	77.8
200	1120	32	M	28.7	42.1	30.5	37.0	28.6	42.9	26.1	42.3
400	1227	16	M	121.0	167.0	129.2	156.6	138.1	147.6	136.4	151.4
400	1227	32	M	59.5	86.2	64.1	78.0	64.6	81.8	63.3	80.4
400	2283	16	M	117.4	161.5	124.1	149.9	131.3	146.0	127.3	149.6
400	2283	32	M	56.3	83.9	62.1	74.0	63.0	76.9	59.6	78.0

Table 3.3. Average execution times (in seconds) of KL-RB, KL-PM, SA and MFA heuristics, for randomly generated mapping problem instances.

PROBLEM SIZE				AVERAGE EXECUTION TIMES			
N	$ E $	K	T	KL-RB	KL-PM	SA	MFA
200	544	8	H	1.07	5.74	80.72	19.57
200	544	16	H	1.53	13.70	127.17	46.17
200	544	32	H	3.29	29.60	245.10	101.84
200	1120	8	H	1.63	7.61	64.10	14.39
200	1120	16	H	2.2	14.56	144.04	58.11
200	1120	32	H	5.11	40.54	282.65	200.53
200	2152	8	H	2.52	10.93	64.22	26.07
200	2152	16	H	3.46	23.66	156.65	61.94
200	2152	32	H	7.60	45.38	373.85	294.94
400	1227	8	H	2.17	10.05	168.86	25.14
400	1227	16	H	2.98	29.74	310.68	164.17
400	1227	32	H	6.41	68.04	681.10	360.40
400	2283	8	H	3.25	16.02	167.07	26.67
400	2283	16	H	4.36	39.79	383.20	88.61
400	2283	32	H	8.61	88.85	632.80	221.60
400	4298	8	H	5.42	25.49	155.25	90.42
400	4298	16	H	7.05	64.88	402.95	171.26
400	4298	32	H	12.59	125.14	553.00	437.62
200	544	16	M	1.5	1.4	165.7	24.8
200	544	32	M	3.3	29.6	258.7	82.6
200	1120	16	M	2.3	14.8	124.2	36.2
200	1120	32	M	5.6	38.4	293.1	122.0
400	1227	16	M	3.1	26.7	280.5	108.0
400	1227	32	M	6.7	60.4	565.1	375.2
400	2283	16	M	4.4	41.7	363.8	130.9
400	2283	32	M	8.7	82.8	573.5	540.8

Tables 3.1 and 3.2 illustrate the quality of the solutions obtained by KL-RB, KL-PM, SA and MFA heuristics. Average total communication costs of the solutions are displayed in Table 3.1, and average computational loads of the maximum and minimum loaded processors are displayed in Table 3.2. As is seen in Tables 3.1 and 3.2, the quality of the solutions obtained by MFA and SA heuristics are superior to KL heuristic. Solutions found by SA are slightly better compared with the solutions found by MFA, in general. However, in some cases MFA performs better. The total communication costs found by KL-RB is less than the total communication costs found by KL-PM, however load balance of the solutions found by KL-PM is better than KL-RB.

Table 3.3 displays the average execution times of KL-RB, KL-PM, SA and MFA heuristics, for the generated mapping problem instances. As is expected, KL heuristic is faster compared with MFA and SA heuristics. Observe that, MFA is always faster than SA. Execution time of MFA is comparable to KL-PM whereas, KL-RB is significantly faster compared with MFA and KL-PM. However, MFA is expected to perform better if an efficient cooling schedule can be devised by analyzing the algorithm in detail, which still remains as an open research issue. Furthermore, the execution times displayed in Table 3.3 for MFA are not obtained by running the most efficient implementation proposed in Section 3.3.2. The time complexity of the implemented scheme is $O(d_{avg} \times K^2)$ whereas the complexity of the most efficient scheme proposed in Section 3.3.2 is $O(d_{avg} \times K + K^2)$. Hence, the execution time of the algorithm is expected to decrease significantly for large d_{avg} and K .

3.5 Parallelization of Mean Field Annealing Algorithm

As is mentioned earlier, heuristic algorithm used for solving the mapping problem is a preprocessing overhead introduced for the efficient implementation of a given parallel program on the target multicomputer. If the mapping heuristic is implemented sequentially, this preprocessing can be considered as the serial portion of the parallel program which limits the maximum efficiency of the parallel program on the target machine. For a fixed parallel program instance,

the execution time of the parallel program is expected to decrease with increasing number of processors in the target multicomputer. However, as is seen in Table 3.3, for a fixed TIG, the execution time of all mapping heuristics increase with increasing number of processors in the target multicomputer. Hence, the serial fraction of the parallel program will increase with increasing number of processors. Thus, this preprocessing will begin to constitute a drastic limit on the maximum efficiency of the overall parallelization due to Amdahl's Law. Hence, parallelization of these mapping heuristics on the target multicomputer is a crucial issue for efficient parallel implementations.

Unfortunately, parallelization of the mapping heuristics introduces another mapping problem. The computations of the mapping heuristics should be mapped to the processors of the same target architecture. However, in this case, the parallel algorithm for the mapping heuristic should be such that its mapping can be achieved *intuitively*. Furthermore, the intuitive mapping should lead to an efficient parallel implementation of the mapping heuristic. For these reasons, the target mapping heuristic to be parallelized should involve regular and inherently parallel computations. MFA algorithm proposed in Section 3.3 for the general mapping problem has these properties for efficient parallelization. Following paragraphs discuss the efficient parallelization of the proposed mapping heuristic for multicomputers.

Assume that, MFA heuristic is to be used to map a given parallel program represented with a TIG having N vertices on a target multicomputer with K processors. The MFA heuristic will use an $N \times K$ spin matrix for the mapping operation. The question is to map the computations of the MFA heuristic to the same target computer (with the same number of K processors). As is mentioned earlier, MFA heuristic is an iterative algorithm. Hence, the mapping scheme can be devised by analyzing the computations involved in a particular iteration of the algorithm. Atomic task can be considered as the computations required for updating an individual spin. Note that, K spin averages at a particular row of the spin matrix are updated at each iteration. Hence, these K spin updates can be computed in parallel by mapping each spin in a row of the spin matrix to a distinct processor of the target architecture. Thus, the $N \times K$ spin matrix is partitioned column-wise such that each processor

is assigned an individual column of the spin matrix. That is, column p of the spin matrix is mapped to processor p of the target architecture. Each processor is held responsible for maintaining and updating the spin values in its local column. Assume that, task- i is selected at random in a particular iteration. Then, each processor is responsible for updating the probability of task i being mapped to itself.

A single iteration of the MFA algorithm can be considered as a three phase process, namely, mean field computation phase, spin update phase, and energy difference computation phase. Each processor p should compute its mean field ϕ_{ip} (Eq. (3.5) or Eq. (3.10)) in the first phase, in order to update its local spin s_{ip} (Eq. (3.6)) by using this mean field value in the second phase. As is mentioned earlier, mean field computation phase is the most time consuming phase of the MFA algorithm. Fortunately, mean field computations are inherently parallel since there is no interactions between mean field computations involved in a particular iteration. However, a close look to Eq. (3.5) and Eq. (3.10) reveals that each processor needs most recently updated values of all spins except the ones in the i -th row in order to compute its local mean field value. Recall that, each processor maintains only a single column of updated spin values due to the proposed mapping scheme. Hence, this computational interaction necessitates *global* interprocessor communication just prior to the distributed mean field computation at each iteration. The volume of global interprocessor communication is proportional to $O(N \times K)$, since each processor p needs all updated spin values except the ones in the i -th row, in order to compute its local ϕ_{ip} . The volume of global interprocessor communication can be reduced to $O(K)$ by considering the parallelization of the matrix equation given in Eq. (3.14).

Eq. (3.14) involves the following operations : construction of the Λ_i and Ψ_i vectors, dense matrix vector product $\Theta_i = \mathbf{D} \times \Lambda_i$ and vector addition $\Phi_i = -\Theta_i - r\Psi_i$. Note that, each processor p only needs to compute the p -th entry θ_{ip} of the Θ_i vector, and the p -th entry ψ_{ip} of the Ψ_i vector in order to compute its local mean field value ϕ_{ip} in parallel. The matrix vector product can be performed in parallel by employing the *scalar accumulation* (SA-MVP) scheme. In this scheme, each processor needs only the p -th row \mathbf{d}_p of the dense

\mathbf{D} matrix and the whole column vector $\Lambda_{\mathbf{i}}$.

Each processor p can concurrently compute the p -th entry λ_{ip} of the $\Lambda_{\mathbf{i}}$ vector by using Eq. (3.12). Note that, q in Eq. (3.12) should be replaced by p in these computations. Then, a global collect (GCOL) operation is required for each processor to obtain a local copy of the $\Lambda_{\mathbf{i}}$ vector. The GCOL operation is essentially appending K local scalars, in order, into a vector of size K and then duplicating this vector in the local memory of each processor. The GCOL operation requires global interprocessor communication. Note that, only K local spin values should be collected globally thus reducing the volume of communication during the GCOL operation by an asymptotic factor of N .

After the GCOL operation, each processor has a local copy of the global $\Lambda_{\mathbf{i}}$ vector. Hence, each processor p can concurrently compute its local θ_{ip} by performing the inner-product $\theta_{ip} = \mathbf{d}_p \times \Lambda_{\mathbf{i}}$. Then, each processor p should compute the p -th entry ψ_{ip} of the $\Psi_{\mathbf{i}}$ vector. Note that, each processor p already maintains the γ_p^{old} value. Hence, each processor can concurrently compute ψ_{ip} using Eq. (3.18). Then, each processor p can concurrently compute its local mean field value ϕ_{ip} by performing the local computation $\phi_{ip} = -\theta_{ip} - r\psi_{ip}$. Note that, these computations are completely local computations and involves no interprocessor communication.

The second phase of an individual iteration of the MFA algorithm is highly sequential since global interaction exists between spin updates due to the normalization process indicated by Eq. (3.6). Fortunately, the strong interaction can be relieved by noting the independent exponentiation operations involved in the numerator of Eq. (3.6). Hence, each processor p can concurrently compute its local $e^{\psi_{ip}/T}$ values. Then, a global sum (GSUM) operation is required for each processor to obtain a local copy of the global sum of the local exponentiation results. The GSUM operation requires *global* interprocessor communication. After the GSUM operation each processor p can concurrently update its local spin value by computing Eq. (3.6). After computing s_{ip}^{new} , each processor p should concurrently update its local γ_p values by using Eq. (3.18) for the use in the next iteration.

In the third phase, each processor should compute the same local copy of

the global energy difference ΔH_i for global termination detection. Each processor p can concurrently compute its local energy difference $\Delta H_{ip} = \phi_{ip} \Delta s_{ip} = \phi_{ip}(s_{ip}^{new} - s_{ip}^{old})$ due to its local spin update. Then, a GSUM operation, which requires global interprocessor communication, is required for each processor to compute a local copy of the global sum $\Delta H_i = \sum_{p=1}^K \Delta H_{ip}$.

Hence, the proposed parallel MFA algorithm necessitates three global communication operations due to the GCOL operation involved during the first phase and two GSUM operations involved in the second and third phases. In fine grain multicomputers, the volume of interprocessor communication is the important factor in predicting the complexity of the interprocessor communication overhead. However, in medium grain multicomputers the number of communications is also important since high *set-up* time overhead is associated with each communication step. For example, set-up time is the dominating factor for short messages in such architectures. Note that, only a single floating variable representing the running sum, is communicated during the GSUM operations involved in the last two phases of the parallel MFA algorithm.

Hence, reducing the number of GSUM operations required in the MFA algorithm will be a valuable asset in achieving efficient implementations on medium grain multicomputers. As seen in Eq. (3.9), there is an execution dependency between the computation of the energy difference ΔH_i and spin-row updates. This execution dependency between the second and the third phase computations can be relieved by rewriting the expression for ΔH_i as follows

$$\begin{aligned}
 \Delta H_i &= \sum_{p=1}^K \phi_{ip}(s_{ip}^{new} - s_{ip}^{old}) \\
 &= \sum_{p=1}^K \phi_{ip}s_{ip}^{new} - \sum_{p=1}^K \phi_{ip}s_{ip}^{old} \\
 &= H_i^{new} - H_i^{old}
 \end{aligned} \tag{3.21}$$

where $H_i = \sum_{p=1}^K \phi_{ip}s_{ip}$ is the *partial* energy contribution to the total energy H due to the spin values at the i -th row (i.e. $H = \sum_{i=1}^N H_i$). The expression for the partial energy H_i can be expanded as

$$H_i = \sum_{p=1}^K \phi_{ip}s_{ip} = \sum_{p=1}^K \phi_{ip} \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^K e^{\phi_{iq}/T}}$$

$$= \frac{1}{A_i} \sum_{p=1}^K \phi_{ip} e^{\phi_{ip}/T} = \frac{B_i}{A_i} \quad (3.22)$$

where $A_i = \sum_{p=1}^K e^{\phi_{ip}/T} = \sum_{p=1}^K a_{ip}$ and $B_i = \sum_{p=1}^K \phi_{ip} e^{\phi_{ip}/T} = \sum_{p=1}^K b_{ip}$.

Hence, after each processor p computes its local $a_{ip} = e^{\phi_{ip}/T}$ and $b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$ values, two global summations $A_i = \sum_{p=1}^K a_{ip}$ and $B_i = \sum_{p=1}^K b_{ip}$ can be accumulated in a *single* GSUM operation. After this single GSUM operation, each processor p can concurrently update its local spin value and compute its new partial energy value as $s_{ip} = a_{ip}/A_i$ and $H_i^{new} = B_i/A_i$. If each processor keeps the partial energy H_i^{old} associated with each row then each processor may concurrently compute the same local copy of the global total energy difference $\Delta H = \Delta H_i = H_i^{new} - H_i^{old}$. Note that, this scheme reduces the number of GSUM operation from two to one. However, the volume of interprocessor communication remains the same since two floating point variables, representing the running sums A_i and B_i , are communicated during the communication steps involved in the GSUM operation.

The node program for a single iteration of the parallel MFA algorithm proposed for solving the mapping problem is given in Figure 3.3. Note that, variables with “ ip ” and “ p ” subscripts denote the local variables. Variables with “ i ” subscripts denote the global variables which are constructed and duplicated at the local memory of each processor after performing the indicated global operations. The proposed parallel algorithm can easily be implemented on any multicomputer having the GCOL and GSUM facilities.

As is seen in Figure 3.3, the proposed parallel MFA algorithm achieves perfect load balance. The parallel computational complexity of a single MFA iteration can be obtained as follows. During the parallel computation of λ_{ip} ’s (step 2) each processor performs $N - 1$ ($d_i - 1$) multiplication/addition operations for dense (sparse) TIGs. Here, d_i denotes the degree of vertex i in the TIG. During the parallel SA-MVP computation (step 3) each processor performs K multiplication/addition operations for both dense and sparse TIGs since the \mathbf{D} matrix is a dense matrix. Each processor performs the same constant amount of arithmetic operations in the remaining steps (steps 5-7 and

-
1. Select a task i at random.
 2. Compute $\lambda_{ip} = \sum_{j \in \text{Adj}(i)} e_{ij} s_{jp}$
 3. Perform GCOL operations to obtain a local copy of

$$\Lambda_i = [\lambda_{i1}, \dots, \lambda_{ip}, \dots, \lambda_{iK}]^T$$
 4. Compute the inner product $\theta_{ip} = \mathbf{d}_p^T \times \Lambda_i$
 5. Compute $\psi_{ip} = w_i(\gamma_p - w_i s_{ip})$
 6. Compute the local mean field value $\phi_{ip} = \theta_{ip} + r\psi_{ip}$
 7. Compute $a_{ip} = e^{\phi_{ip}/T}$ and $b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$
 8. Perform GSUM to compute the local copies of

$$A_i = \sum_{p=1}^K a_{ip} \quad \text{and} \quad B_i = \sum_{p=1}^K b_{ip}$$
 9. Compute $s_{ip}^{\text{new}} = a_{ip}/A_i$ and then $\Delta s_{ip} = s_{ip}^{\text{new}} - s_{ip}^{\text{old}}$
 10. Compute $H_i^{\text{new}} = B_i/A_i$ and then $\Delta H_i = H_i^{\text{new}} - H_i$
 11. Update $\gamma_p = \gamma_p + w_i \Delta s_{ip}$
 12. Update $s_{ip} = s_{ip}^{\text{new}}$ and $H_i = H_i^{\text{new}}$
-

Figure 3.3. Node program for one iteration of the parallel MFA algorithm for the mapping problem.

steps 7-12). Hence, the parallel computational complexity of the proposed algorithm is $O(N + K)$ and $O(d_{avg} + K)$ for dense and sparse TICs respectively. Hence, linear speed-up can easily be achieved if communication overhead remains negligible. The communication complexity due to the GCOL (step 3) and GSUM (step 8) operations are discussed in general in the following paragraph.

The interconnection schemes used in the processor organization of the multicomputers are usually symmetric in nature (i.e. POG is symmetric). GSUM and GCOL type of operations in such architectures is performed in two phase. In the first phase, a sequence of concurrent single-hop communications are performed to accumulate or collect the result in a root processor. In the second phase, the final result is broadcast from this root processor again using a sequence of concurrent single-hop communications. The number of concurrent single-hop communications in each phase will be proportional to diameter of the POG. For example, diameters of hypercube and mesh POGs are $\log_2 K$ and $K^{1/2}$, respectively. The overall concurrent volume of communications will be proportional to diameter and number of processors (K) in both phases of the GSUM and GCOL operations, respectively. If a full-duplex pair of communication links are used between each pair of directly connected processors (e.g. Intel's iPSC/2) then, such global operations are performed in single phase by a sequence of concurrent single-hop exchange communications. In such an architecture, the number of concurrent single-hop communications and the overall volume of concurrent communication in GSUM and GCOL operations can be reduced by a factor of two.

4. MFA FOR THE CIRCUIT PARTITIONING PROBLEM

This chapter presents formulation of Mean Field Annealing (MFA) for solving the circuit partitioning problem. Section 4.1 describes the circuit partitioning problem, and summarizes the previous works on the circuit partitioning problem. In Section 4.2 the circuit partitioning problem is modeled as the graph partitioning problem and the network partitioning problem. Section 4.3 presents the formulation of MFA for the graph partitioning problem and the network partitioning problem. MFA algorithms proposed for solving the graph partitioning problem and the network partitioning problem are parallelized as is described in Section 4.4.

4.1 The Circuit Partitioning Problem

Partitioning of a VLSI circuit, which is defined with its components and signal nets, is an extensively studied problem. Partitioning means to divide the components of a circuit into two or more evenly weighted partitions, such that the number of signal nets interconnecting them is minimized. This problem, called the circuit partitioning problem, arises while dividing a circuit into parts that will be implemented separately. In some layout problems like placement and floor-planning, divide-and-conquer algorithms, which necessitate dividing up the circuits hierarchically into parts with different minimization criteria, are used. Circuit partitioning is also needed within these algorithms [20]. The circuit partitioning problem first appeared because of the need for partitioning components of electronic circuits to circuit boards, minimizing the connections

between boards. A heuristic for solving this problem is given in the seminal paper by Kernighan and Lin [17]. In this work, the circuits are represented as graphs and the problem is treated as the graph partitioning problem. In a later work by Schweikert and Kernighan [27, 37], deficiencies of using graph model are illustrated, and a new model called net-cut circuit model is proposed. The problem of partitioning circuits using this representation is called the network partitioning problem.

As both of the mentioned problems (the graph partitioning problem and the network partitioning problem) are proved to be NP-hard [8], finding efficient heuristics for them is an important issue. Various heuristics, e.g., Kernighan-Lin like algorithms [7], Simulated Annealing (SA) etc., are proposed and implemented for solving these problems [20]. In this chapter, Mean Field Annealing (MFA) algorithm, is formulated for the circuit partitioning problem.

Algorithms used for solving the circuit partitioning problem are time consuming processes, and parallelization of them is crucial. In this chapter, parallelization of MFA algorithms for solving the circuit partitioning problem on distributed-memory, message-passing multicomputers is also addressed.

4.2 Modeling the Circuit Partitioning Problem

An instance of the circuit partitioning problem constitutes of a set of weighted components and a list of nets which defines the connection relationships among these components. Nets can also be weighted; but, as this does not change the nature of the problem, we assume the weights of the nets to be unity. An example instance of the circuit partitioning problem is given below.

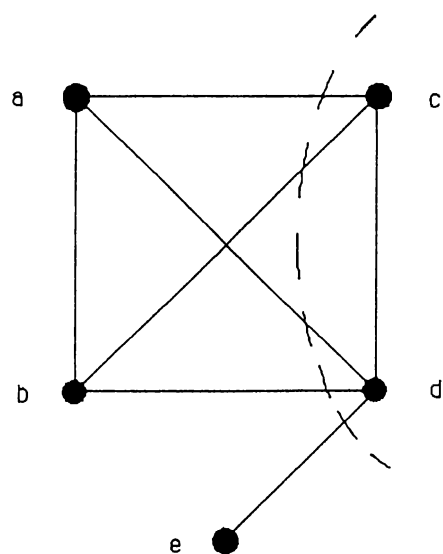
components	weights	net-list
a	1	net 1 : a-b-c-d
b	1	net 2 : d-e
c	2	
d	1	
e	1	

The problem is to divide the given circuit into M ($M \geq 2$) equally weighted partitions, while minimizing the number of *external* connections among partitions. In Schweikert and Kernighan algorithm [27, 37], external lines are reduced based on the following criteria

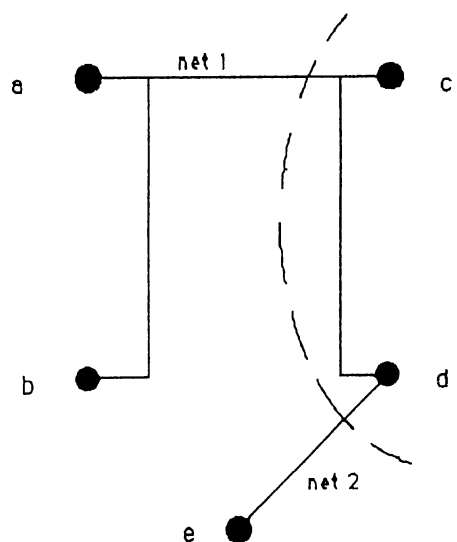
- 1) When all components of the same net are in the same block, moving any one of the components to another block will create an additional external line.
- 2) If a net has all its components in a block except one component, moving that component to the same block will remove the net from the cut.
- 3) If components of a net are in more than one block, number of external connections does not change by moving components of the net within blocks, if the number of blocks that the net is distributed does not change.

In order to transform the given circuit partitioning problem instance to a graph partitioning problem instance, each net is represented by a clique of its terminals. Resulting graph instance is shown in Figure 4.1(a). Observe that this representation changes the structure of the connections in the given circuit. Representation of the given instance as a network is given in Figure 4.1(b). A network consists of a set of components called cells and a set of signal nets (or only nets). A net is a subset of the set of cells. This representation exactly simulates the connection relationships among components.

In order to show the deficiency of the graph model, the partitions indicated with dashed lines in Figure 4.1 will be examined. Observe that, in Figure 4.1(a), cut size is equal to 5. In Figure 4.1(b), it is 2, which is the actual cut size. The cost contribution of a unit cost net across a cut of a bipartition is 1. The cost contribution of a clique, that is evenly split across a cut, rises quadratically with the size of the clique. This quadratic growth does not adequately reflect the costs arising in practice [20]. Although there can be some attempts to solve this dilemma, there is no good way of mapping a circuit instance into a graph [20].



(a)



(b)

Figure 4.1. Modeling of a given circuit partitioning problem instance with (a) graph and (b) network models. Dashed lines indicate an example partition.

4.3 Solving the Circuit Partitioning Problem Using MFA

In this section, formulation of MFA for the circuit partitioning problem, using two different models is given. Graph and network models are described in the following two sections respectively.

4.3.1 Graph Model

If the graph model is used for the representation of the circuit partitioning problem, the problem can be treated as the graph partitioning problem. A formal definition of the graph partitioning problem is as follows: A graph $G = (V, E)$ with $|V| = N$ vertices $(1, 2, \dots, i, j, \dots, N)$, vertex weights $(w_1, w_2, \dots, w_i, w_j, \dots, w_N)$, and edges E between vertices with weights e_{ij} is given. The question is to divide the graph into M partitions of nearly equal weights such that the *cut size* is minimized.

Similar formulations of MFA for partitioning fully connected graphs are given in [4, 21, 35]. However, graphs arising in circuit partitioning are usually sparse. In order to avoid redundant computation, the algorithm is modified to work for sparse graphs. As in the previous works [4, 21, 35], a spin (i.e. neuron) matrix which consists of N vertex-rows and M partition-columns is used as a representation scheme. The output s_{ip} of a spin (i, p) denotes the probability of finding vertex i in partition p ($1 \leq p \leq M$).

We propose the following energy function for sparse graphs, where $Adj(i)$ denotes the set of vertices connected to vertex i .

$$H(s) = \frac{1}{2} \sum_{i=1}^N \sum_{j \in Adj(i)} \sum_{p=1}^M e_{ij} s_{ip} (1 - s_{jp}) + \frac{r}{2} \sum_{p=1}^M \sum_{i=1}^N \sum_{j \neq i} s_{ip} s_{jp} w_i w_j \quad (4.1)$$

Here, $(1 - s_{jp})$ denotes the probability of vertex j being in a partition other than partition p . Hence, $s_{ip} \times (1 - s_{jp})$ denotes the probability of vertex i being in partition p and vertex j in a different partition. Then, term $e_{ij} \times s_{ip} \times (1 - s_{jp})$ denotes the cost contribution of edge (i, j) to the cut size by mapping vertices i and j to different partitions. As the first summation term in

Eq. (4.1) covers all vertices and all partitions, it represents the total cut size of a partitioning represented by the values of the spins in the spin matrix. Hence, this summation term is used for minimizing the weighted sum of edges on the *cut*. Second triple summation term in Eq. (4.1) computes the summation of the inner products of the weights of the vertices in each partition. This term will have the global minimum value only when the summations of the weights of the vertices in each partition are equal. The parameter r in Eq. (4.1) is introduced to maintain a balance between the two optimization objectives of the original graph partitioning problem.

Using the mean field approximation given in Eq. (2.8), *mean field* of a spin (i, p) for the energy function defined in (4.1) can be computed as

$$\phi_{ip} = - \sum_{j \in Adj(i)} e_{ij}(1 - s_{jp}) - r \sum_{j \neq i}^N s_{jp} w_i w_j \quad (4.2)$$

In this equation, first summation term shows the rate of increase in the cut size by placing vertex i in partition p . Second summation term shows the rate of increase in the cost term, introduced for balancing the partitions, by placing vertex i in partition p .

The probability that vertex i is in partition p is then normalized as follows:

$$s_{ip} = \frac{e^{\phi_{ip}/T}}{\sum_{q=1}^M e^{\phi_{iq}/T}} \quad (4.3)$$

Note that, this normalization guarantees that each vertex is included in only one partition.

MFA algorithm for the graph partitioning problem is similar to MFA algorithm for the mapping problem, which is described in the Chapter 3, except mean field computations. Mean fields of spins are computed using Eq. (4.2) in MFA algorithm for the graph partitioning problem. Note that, second term in Eq. (4.2) is same as the second term in the mean field equation of the MFA algorithm for the mapping problem (Eq. (3.5)). Hence, this term can be computed in constant time ($O(1)$), for each mean field computation, as described in Section 3.3.2 by defining γ_p as

$$\gamma_p = \sum_{j=1}^N w_j s_{jp} \quad (4.4)$$

Then, Eq. (4.2) can be rewritten as

$$\phi_{ip} = - \sum_{j \in Adj(i)} e_{ij}(1 - s_{jp}) - rw_i \gamma_p - w_i^2 s_{ip} \quad (4.5)$$

Note that, γ_p represents weighted sum of spin values of the p -th column of the spin matrix. Hence, initial γ_p value of each column p ($1 \leq p \leq M$) can be computed by using Eq. (4.4) for the initial spin values. Then, γ_p values can be updated at the end of each iteration (i.e. after spin updates) by using

$$\gamma_p^{new} = \gamma_p^{old} - w_i s_{ip}^{old} + w_i s_{ip}^{new} \quad (4.6)$$

for $1 \leq p \leq M$.

Computation of the first term in Eq. (4.2) is $O(d_{avg})$ where, d_{avg} denotes the average degree of the vertices of the graph $G(V, E)$. Then, complexity of mean field computations for a spin row is $O(M \times (d_{avg} + 1)) = O(M \times d_{avg})$. Complexity of spin update computations and energy difference computation performed at each iteration of the MFA algorithm are both $O(M)$. Hence, the overall complexity of a single MFA iteration for the graph partitioning problem is $O(M \times d_{avg})$.

Performance of the MFA algorithm for solving the graph partitioning problem in comparison with SA and Kernighan-Lin heuristics is extensively studied in [21, 35]. Results obtained using MFA are very encouraging, comparable to results obtained by SA and Kernighan-Lin heuristics.

4.3.2 Network Model

In this section, a suitable mapping of MFA to the network partitioning problem is proposed. With this mapping, disadvantages of using graph model to represent a circuit partitioning problem instance are avoided. Following is a formal definition of the network partitioning problem. A network with N cells $(1, 2, \dots, i, j, \dots, N)$, cell weights $(w_1, w_2, \dots, w_i, w_j, \dots, w_N)$, and a list of *nets* (n_1, n_2, \dots) , with weights $(w_{n_1}, w_{n_2}, \dots)$ is given. The question is to partition the network into M partitions of nearly equal weights such that the *cut size* is minimized.

Following energy function is proposed for the network partitioning problem

$$\begin{aligned}
 H(s) = & \frac{1}{2} \sum_{i=1}^N \sum_{p=1}^M \sum_{q \neq p} \sum_{n \in N_i} \max\{s_{jq(j \in n)}\} s_{ip} w_n \\
 & + \frac{r}{2} \sum_{p=1}^M \sum_{i=1}^N \sum_{j \neq i}^N s_{ip} s_{jp} w_i w_j
 \end{aligned} \tag{4.7}$$

where N_i denotes the set of nets connected to cell i , and $\max(S)$ denotes the maximum value in set S . In Eq. (4.7), $s_{jq(j \in n)}$ indicates the set of spin values which denote the probabilities of finding the cells $j \in n$ (cells belonging to the net n), in partition q . Hence, $\max\{s_{jq(j \in n)}\}$ denotes the maximum spin value among the indicated set of spin values. Then, term $\max\{s_{jq(j \in n)}\} \times s_{ip} \times w_n$ shows the cost contribution of net n to the cut size by putting cell i in partition p and at least one of the cells in net n to another partition. With these observations it can be seen that first summation term in Eq. (4.7) represents the total cut size caused by the nets whose cells are in more than one partitions. Second summation term in Eq. (4.7) is same as the second summation term in Eq. (4.1), and maintains the weight balance among partitions.

As described in Chapter 2 mean field of a spin is calculated by taking the partial derivative of the energy function with respect to the expected value of that spin. Energy function defined by Eq. (4.7) is not differentiable because of the $\max()$ function. If the mean field of a spin is interpreted intuitively as the effect of the values of the other spins to the value of that spin, then mean field of a spin (i, p) due to Eq. (4.7) can be written as

$$\phi_{ip} = - \sum_{q \neq p}^M \sum_{n \in N_i} \max\{s_{jq(j \in n)}\} w_n - r \sum_{j \neq i}^N s_{jp} w_i w_j \tag{4.8}$$

Note that, in this equation first term shows the rate of increase in the cut size by placing vertex i in partition p . Second summation term is similar to the term in Eq. (4.2) and has the same meaning as described above.

The normalization operation (i.e. normalization of the spin values) remains same as in the formulation of the graph partitioning problem.

Three MFA algorithms given for the mapping problem, the graph partitioning problem and the network partitioning problem are same except the mean field computations, which constitute the problem specific part of the

MFA algorithms. Mean field computations in the MFA algorithm for the network partitioning problem are performed using Eq. (4.8). Second term in Eq. (4.8) is computed efficiently in constant time for each mean field computation as described in the previous section for the graph partitioning problem. Observe that complexity of computing the first term in Eq. (4.8) is $O(M \times c \times (s - 1)) = O(M \times c \times s)$ for each mean field computation, where M is the number of partitions, c is the average number of nets that a cell is connected, and s is the average size of a net (size of a net is the number of cells in a net). Note that, $c \times (s - 1)$ corresponds to average degree of a vertex in the graph model (i.e., $c \times (s - 1) = d_{avg}$). At each iteration of the MFA algorithm M spins are updated, hence, M mean field computations are performed. Then, complexity of mean field computations in a single iteration of the MFA algorithm is $O(M \times (M \times c \times s + 1)) = O(M^2 \times c \times s)$. However, this complexity can be reduced using the following observation. Eq. (4.8) can be rewritten as

$$\begin{aligned} \phi_{ip} &= -\left(\sum_{q=1}^M \sum_{n \in N_i} \max\{s_{jq(j \in n)}\} w_n - \sum_{n \in N_i} \max\{s_{jp(j \in n)}\} w_n\right) \\ &\quad - r \sum_{j \neq i}^N s_{jp} w_i w_j \\ &= -(\psi_i - \psi_{ip}) - r \sum_{j \neq i}^N s_{jp} w_i w_j \end{aligned} \quad (4.9)$$

where

$$\psi_i = \sum_{q=1}^M \sum_{n \in N_i} \max\{s_{jq(j \in n)}\} w_n \quad (4.10)$$

$$\psi_{ip} = \sum_{n \in N_i} \max\{s_{jp, (j \in n)}\} w_n \quad (4.11)$$

Values ψ_i and ψ_{ip} given in Eq. (4.10) and Eq. (4.11) can be computed together in $O(M \times c \times s)$ at the beginning of each iteration of the MFA algorithm. Hence, complexity of mean field computations for a spin row is $O(M \times c \times s + M) = O(M \times c \times s)$. Complexity of spin update computations and energy difference computation performed at each iteration of the MFA algorithm are both $O(M)$. Then the complexity of one iteration of the MFA algorithm for the network partitioning problem is $O(M \times c \times s)$.

In order to demonstrate the effectiveness of the network model, the behavior of the energy function defined in MFA will be examined. Two possible solutions

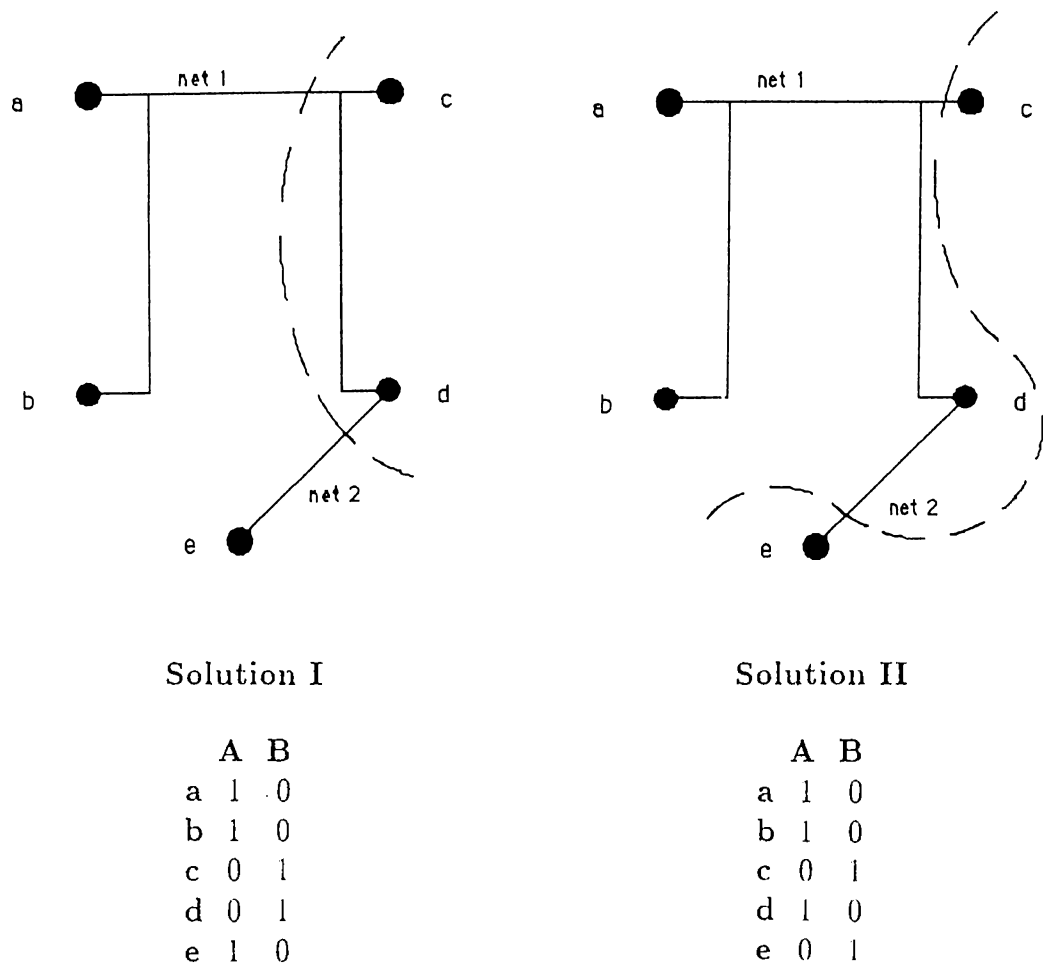


Figure 4.2. Two possible solutions for the given circuit partitioning problem instance.

to the instance given in Section 4.2 are illustrated in Figure 4.2 as $A = \{a, b, c\}$, $B = \{c, d\}$ and $A = \{a, b, d\}$, $B = \{c, e\}$, where A and B denote the two partitions. Neuron matrix representation of these solutions are also given in Figure 4.2 using a 5×2 spin matrix.

The *energy* values of the two states of the spin matrix defined by Solution 1 and 2 are computed for the graph model (using Eq. (4.1)) as $H_1 = A \times 5 + 5$ and $H_2 = A \times 4 + 5$ respectively. The *energy* values computed for the network model (using Eq. (4.7)) are $H_1 = H_2 = A \times 3 + 5$. In graph model, second solution is favored more than the first solution; but, it can be seen that the actual cut sizes are equal in both solutions. So, in graph model, some solutions are favored to other ones although they have the same quality, meaning that some features of the circuit partitioning problem is not represented correctly. However, in network model *energies* of the two solutions are the same $H_1 = H_2$, which gives the desired result. Hence, it can be concluded that network model is a better scheme for mapping the circuit partitioning problem to MFA.

The performance of the proposed MFA algorithm for solving the network partitioning problem is demonstrated in Table 4.1 for three different problem sizes. MFA is compared with SA and Kernighan-Lin (KL) heuristics. An efficient variation of Kernighan-Lin heuristic [7] which is proposed for network partitioning is implemented. These heuristics are tested for randomly generated networks with various number of cells (N) and nets (L), and maximum net sizes (S). In these networks, weights of the cells and nets are taken to be unity. Networks are partitioned into two bins, and balance criteria of the heuristics are set such that differences between the weights of the resulting bins were less than % 5 of the total weights of the cells. As seen in the table, performance of MFA is close to SA, and better than KL in some instances. Execution time of SA is maximum, 120 times that of KL on the average. MFA is, 60-70 times slower than KL and 2 times faster than SA. Time complexity of the MFA algorithm used in these experiments was $O(M^2 \times c \times s + N \times M)$. In [35], using the notion of critical temperature, better timings of MFA are obtained. Probably, by determining the critical temperature, MFA will run much faster for these instances. KL heuristic is faster compared with the general

Table 4.1. Mean cut sizes of the solutions found by MFA, KL, and SA heuristics for randomly generated network partitioning problem instances.

PROBLEM SIZE			MEAN CUT SIZE		
N	L	S	MFA	SA	KL
128	205	4	75.3	74.8	77.6
128	102	8	52.0	49.2	52.4
128	69	16	44.4	41.5	44.3
256	543	4	217.9	211.0	217.9
256	240	8	126.8	124.7	126.2
256	200	16	139.5	131.4	134.2
512	784	4	272.0	258.0	273.0
512	809	8	477.6	471.0	481.4
512	336	16	215.4	213.6	219.8

heuristics like MFA and SA since it is an efficient, problem specific heuristic, having almost linear time complexity. However, KL heuristic can only be used for partitioning networks having nets with bounded weights. Linear time complexity of KL heuristic can not be preserved for other types of networks. Furthermore, as is described in the following section, MFA algorithm is more suitable for parallelization compared with SA and KL heuristics. Hence, these results demonstrate that the proposed mapping of the MFA to the network partitioning problem is a promising alternative heuristic for solving the network partitioning problem.

4.4 Parallelization of Mean Field Annealing Algorithm

Efficient parallelization of heuristics used for solving the circuit partitioning problem is crucial since the circuits arising in practice are quite large in general. Parallelization schemes for MFA algorithms used for solving the graph partitioning problem and the network partitioning problem are described in the following sections.

4.4.1 Graph Model

For parallelization of the algorithm, columns of the spin matrix are partitioned among processors such that each processor has M/K columns of the spin matrix. Here, K denotes the number of processors in the target multicomputer. Hence, each processor is assigned the data and the computations associated with all N vertices for only M/K partitions. That is, each processor is assigned $N \times M/K$ spins. This decomposition yields perfect load balance if M is a multiple of K or $M \gg K$. Each processor stores its local column slice of the global spin matrix in row-wise order for the sake of efficient access to the spin values. Host processor initializes the spin matrix and sends to the node processors their portions. At each iteration, spin values corresponding to the selected vertex are updated by computing the mean field value of each spin, and difference between new energy and old energy is calculated. If energy difference is less than a predefined constant for a number of subsequent iterations, temperature is decreased, and iteration is started again. Two phases of a MFA iteration (i.e., mean field computations and energy difference calculation) are interleaved as described for the mapping problem in Chapter 3. The parallel algorithm for the node program for a single iteration of MFA algorithm is given in Figure 4.3.

In the parallel MFA algorithm for solving the graph partitioning problem, each processor selects a vertex i at random, where the random sequence in each processor is the same. Hence, no global communication is necessary for broadcasting the selected vertex. Then, each processor computes the mean fields of the randomly selected vertex only for its *local partitions*. After computing mean fields of the local spins two partial summation terms are computed at steps 3 and 4. Then, a global sum (GSUM) operation is performed at step 5 to accumulate the overall summations in each processor. Each processor updates its local spin values at step 6 and computes ΔH_i at step 7. At step 8, γ_p values are updated. Details of the parallel MFA program for solving the graph partitioning problem is given in [4]. Note that, only one global communication is needed at each iteration of the algorithm. As is mentioned in Section 3.5, global communication is performed as a sequence of single-hop exchange communications. Volume of communications at each exchange step is fixed to 2 floating

-
1. Select a vertex i at random.
 2. For each local partition $p := 1$ to M/K compute mean field values

$$\phi_{ip} = -\sum_{j \in Adj(i)} e_{ij}(1 - s_{jp}) - rw_i(\gamma_p - w_i s_{ip})$$

3. For each local partition $p := 1$ to M/K compute

$$a_{ip} = e^{\phi_{ip}/T} \quad \text{and} \quad b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$$

4. Compute partial summations

$$A_i = \sum_{p=1}^{M/K} a_{ip} \quad \text{and} \quad B_i = \sum_{p=1}^{M/K} b_{ip}$$

5. Perform GSUM to compute the local copies of

$$A_i = \sum_{p=1}^K a_{ip} \quad \text{and} \quad B_i = \sum_{p=1}^K b_{ip}$$

6. For each local partition $p := 1$ to M/K compute $s_{ip}^{new} = a_{ip}/A_i$ and

$$\text{then } \Delta s_{ip} = s_{ip}^{new} - s_{ip}^{old}$$

7. Compute $H_i^{new} = B_i/A_i$ and then $\Delta H_i = H_i^{new} - H_i$

8. For each local partition $p := 1$ to M/K update $\gamma_p = \gamma_p + w_i \Delta s_{ip}$

9. For each local partition $p := 1$ to M/K update $s_{ip} = s_{ip}^{new}$ and

$$H_i = H_i^{new}$$

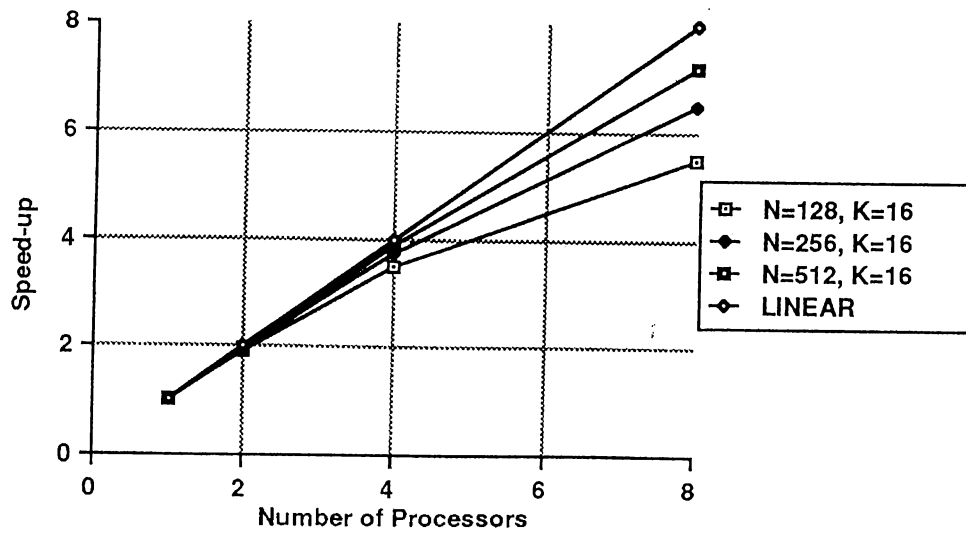
Figure 4.3. Node program for one iteration of the parallel MFA algorithm for the graph partitioning problem.

point words, and does not change with increasing problem size. The number of exchange communication steps in the global summation operation increases with the diameter of the multicomputer. Diameter of a multicomputer implementing hypercube topology is $\log_2 K$ hence, the given parallel algorithm is expected to scale on the hypercube architecture. Figure 4.4 illustrates the speed-up and efficiency curves for the parallel MFA algorithm for solving the graph partitioning problem on a 3-dimensional iPSC/2 hypercube multicomputer for three different problem sizes. As is seen in Figure 4.4, speed-up and efficiency increases with increasing problem size and almost linear speed-up is obtained for large problem sizes.

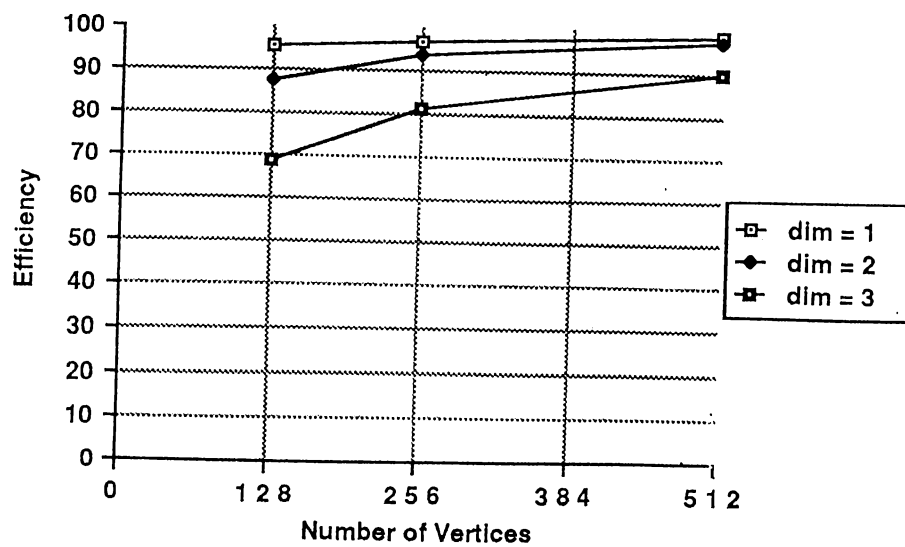
4.4.2 Network Model

Columns of the global spin matrix for the network partitioning problem are partitioned similarly among the processors of the multicomputer, such that each processor is assigned M/K columns of the global spin matrix. As in the graph partitioning problem, host processor initializes the spin matrix and sends to the node processors their portions. Each processor is responsible for the computation of the spin values in its partition. The algorithm for the node program for a single iteration is given in Figure 4.5.

Observe that, there is one more global communication (at step 4) in this algorithm because of the first term in (4.8). The rest of the algorithm is similar to the parallel MFA algorithm for the graph partitioning problem. Although this parallel algorithm requires one more global communication, it is also expected to scale on the hypercube due to its fixed communication requirement (both in number and volume). The speed-up and efficiency curves for the parallel MFA algorithm for the graph partitioning problem on a 3-dimensional iPSC/2 hypercube multicomputer is given in Figure 4.6. As is seen in Figure 4.6, speed-up and efficiency increases as the problem size increases. Almost linear speed-up is obtained for large problem sizes.



(a)



(b)

Figure 4.4. Speed-up (a) and efficiency (b) curves for the graph partitioning problem.

1. Select a cell i at random.

2. For each local partition $p := 1$ to M/K compute

$$\psi_{ip} = \sum_{n \in N_i} \max\{s_{jp}, (j \in n)\} w_n$$

3. Compute partial summation

$$\psi_i = \sum_{q=1}^{M/K} \psi_{ip}$$

4. Perform GSUM to compute the local copies of

$$\psi_i = \sum_{q=1}^M \psi_{ip}$$

5. For each local partition $p := 1$ to M/K compute mean field values

$$\phi_{ip} = -(\psi_i - \psi_{ip}) - rw_i(\gamma_p - w_i s_{ip})$$

6. For each local partition $p := 1$ to M/K compute

$$a_{ip} = e^{\phi_{ip}/T} \quad \text{and} \quad b_{ip} = \phi_{ip} e^{\phi_{ip}/T}$$

7. Compute partial summations

$$A_i = \sum_{p=1}^{M/K} a_{ip} \quad \text{and} \quad B_i = \sum_{p=1}^{M/K} b_{ip}$$

8. Perform GSUM to compute the local copies of

$$A_i = \sum_{p=1}^K a_{ip} \quad \text{and} \quad B_i = \sum_{p=1}^K b_{ip}$$

9. For each local partition $p := 1$ to M/K compute $s_{ip}^{new} = a_{ip}/A_i$ and

$$\text{then } \Delta s_{ip} = s_{ip}^{new} - s_{ip}^{old}$$

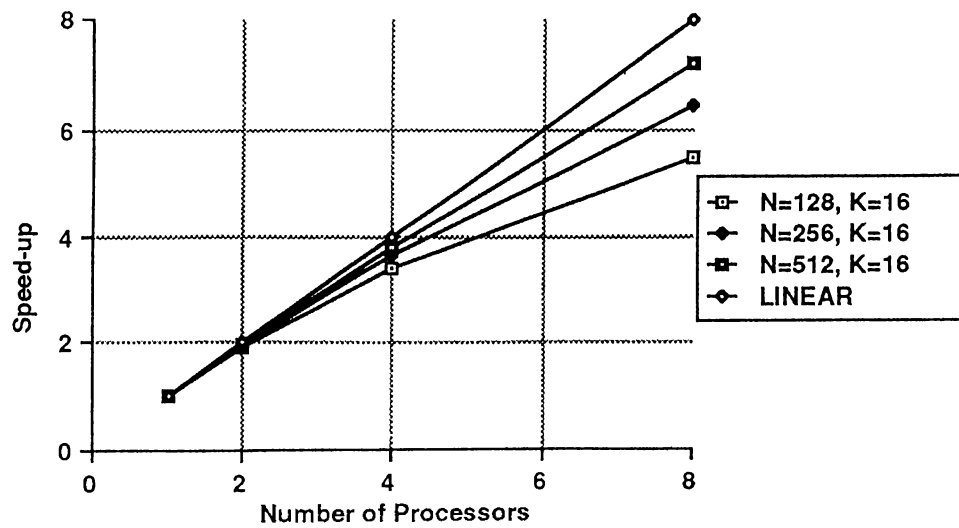
10. Compute $H_i^{new} = B_i/A_i$ and then $\Delta H_i = H_i^{new} - H_i$

11. For each local partition $p := 1$ to M/K update $\gamma_p = \gamma_p + w_i \Delta s_{ip}$

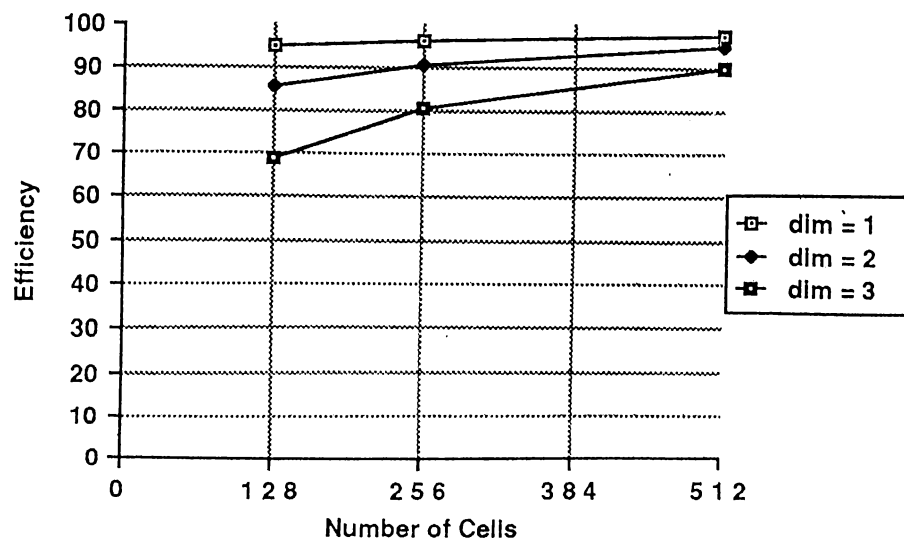
12. For each local partition $p := 1$ to M/K update $s_{ip} = s_{ip}^{new}$ and

$$H_i = H_i^{new}$$

Figure 4.5. Node program for one iteration of the parallel MFA algorithm for the network partitioning problem.



(a)



(b)

Figure 4.6. Speed-up (a) and efficiency (b) curves for the network partitioning problem.

5. CONCLUSIONS

Mean Field Annealing (MFA) algorithm, recently proposed for solving combinatorial optimization problems, combines the characteristics of neural networks and simulated annealing. Previous works on MFA resulted with successful application of the algorithm to some classic optimization problems such as the traveling salesperson problem and the graph partitioning problem. In this work, MFA is formulated for the mapping problem and the circuit partitioning problem. Performances of the proposed heuristics are investigated by comparing them with other well-known heuristics, and efficient parallel versions of the proposed algorithms are developed.

In chapter 2, MFA algorithm is formulated for the mapping problem. An efficient implementation scheme, which decreases the complexity of the proposed algorithm by asymptotical factors, is also given. The performance of the proposed MFA algorithm is evaluated in comparison with two well-known heuristics: simulated annealing and Kernighan-Lin. Algorithms are experimented for a number of randomly generated mapping problem instances. Solution qualities of MFA and simulated annealing heuristics are found to be superior to the efficient Kernighan-Lin heuristic. The solution quality of simulated annealing is slightly better in comparison with MFA whereas, MFA is faster. As is expected, Kernighan-Lin heuristic is faster in comparison with MFA and simulated annealing heuristics. Kernighan-Lin heuristic is faster in comparison with general heuristics as MFA and simulated annealing, since it is an efficient, problem specific heuristic, having linear time complexity. However, linear time complexity of Kernighan-Lin heuristic can not be preserved, if the weights of the edges of the graph to be partitioned are not bounded.

Furthermore, MFA algorithm is more suitable for parallelization in comparison with simulated annealing and Kernighan-Lin heuristics. Hence, obtained results demonstrate that the proposed formulation of the MFA for the mapping problem is a promising alternative heuristic for solving the mapping problem.

Inherent parallelism of the MFA is exploited by designing an efficient parallel algorithm for the proposed MFA heuristic for the mapping problem. Proposed parallel MFA algorithm achieves perfect load balance, and has fixed communication requirement which does not increase with the size of the problem instance.

MFA algorithm is formulated for solving CPP using two alternative models in Chapter 3. It is shown that network model is a better scheme for mapping MFA to the circuit partitioning problem in comparison with the graph model. Performance of the MFA is compared with the performances of Kernighan-Lin and simulated annealing heuristics, using randomly generated circuit partitioning problem instances. Performance of MFA is close to simulated annealing, and better than Kernighan-Lin heuristic in some instances. Execution time of MFA is less than simulated annealing, but more than Kernighan-Lin heuristic. Obtained results indicate that MFA can be used as an alternative heuristic for solving the circuit partitioning problem. MFA algorithms proposed for solving the circuit partitioning problem are parallelized and implemented on an iPSC/2 hypercube multicomputer. Experimental results show that the proposed heuristics can be efficiently parallelized on hypercube multicomputers, which is crucial for algorithms that solve such computationally hard problems.

Results obtained in this work indicates that MFA which is originally proposed for solving the traveling salesperson problem also works for the circuit partitioning problem and the mapping problem, and can be used as a general tool for solving combinatorial optimization problems. Scalability of the algorithm is quite good, reasonable results are obtained for large problem sizes. Performance of the proposed MFA algorithms may be improved by fine tuning of the temperature schedule of the algorithm, which still remains as a research issue.

Inherent parallelism of MFA is exploited in this work by designing efficient

parallel MFA algorithms. Parallelization of heuristics, proposed for solving NP-hard combinatorial optimization problems, is important since the combinatorial optimization problems are computationally hard problems. Development of parallel computers increases the need for heuristics that can be efficiently parallelized. Results obtained in this work show that MFA is a good candidate for developing efficient parallel heuristics. Proposed parallel MFA algorithms are expected to scale on parallel architectures, due to their fixed communication requirements.

Bibliography

- [1] Arora, R. K., and Rana, S. P., "Heuristic algorithms for process assignment in distributed computing systems," *Information Processing Letters*, vol. 11, no. 4-5, pp. 199-203, 1980.
- [2] Bokhari, S. H. "On the mapping problem," *IEEE Trans. Comput.*, vol. 30, no. 3, pp. 207-214, 1981.
- [3] Brandt, R. D., Wang, Y., Laub, A. J., Mitra, S. K. "Alternative Networks for Solving the TSP and the List-Matching Problem," *IEEE Int. Conference on Neural Nets*, Vol.II, pp. 333-340, July 1988.
- [4] Bultan, T., and Aykanat, C. "Parallel mean field algorithms for the solution of combinatorial optimization problems," *Proc. ICANN-91*, vol. 1, pp. 591-596, 1991.
- [5] Bultan, T., and Aykanat, C. "Circuit Partitioning Using Parallel Mean Field Annealing Algorithms," *Proc. 3rd IEEE Symposium on Parallel Processing*, to be published.
- [6] Erçal, F., Ramanujam, J., and Sadayappan, P. "Task allocation onto a hypercube by recursive mincut bipartitioning," *J. Parallel Distrib. Comput.* vol. 10, pp. 35-44, 1990.
- [7] Fiduccia, C. M., and Mattheyses, R. M. "A linear heuristic for improving network partitions," in *Proc. Design Automat. Conf.*, pp. 175-181, 1982.
- [8] Garey, M. R., and Johnson, D. S. *Computers and Intractability*. San Francisco, CA: Freeman, pp. 209-240, 1979.
- [9] Hopfield, J. J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 79,

pp. 2554-2558, 1982.

- [10] Hopfield, J. J. "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 81, pp. 3088-3092, 1984.
- [11] Hopfield, J. J., and Tank, D. W. "'Neural' Computation of¹ Decisions in Optimization Problems," *Biolog. Cybern.*, vol. 52, pp. 141-152, 1985.
- [12] Hopfield, J. J., and Tank, D. W. "Computing with neural circuits: a model," *Science*, Vol. 233, pp. 625-633, August 1986.
- [13] Hopfield, J. J., and Tank, D. W. "Collective computation in neuronlike circuits," *Scientific American*, 257(6):104-114, 1987.
- [14] Hegde, S. U., Sweet, J. L., and Levy, W. B. "Determination of Parameters in a Hopfield/Tank Computational Network," *IEEE Int. Conf. Neural Networks*, vol. 2, pp. 291-298, 1988.
- [15] Indurkha, B., Stone H. S., and Xi-Cheng, L. "Optimal partitioning of randomly generated distributed programs," *IEEE Trans. Software Engrg.*, vol. 12, no. 3, pp. 483-495, 1986.
- [16] Kasahara, H., and Narita, S. "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Trans. Comput.*, vol. 33, no. 11, pp. 1023-1029, 1984.
- [17] Kernighan, B. W., and Lin, S. "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291-307, 1970.
- [18] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [19] Krishnamurthy, B. "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438-446, 1984.
- [20] Lengauer, T. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, pp. 251-258, 1990.

- [21] Peterson, C., and Anderson, J. R. "Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem," *Complex Syst.* vol. 2, pp. 59-89, 1988.
- [22] Peterson, C., and Soderberg, B. "A new method for mapping optimization problems onto neural networks," *Int. J. Neural Syst.*, vol. 1, no. 3, 1989.
- [23] Ramanujam, J., Erçal, F., and Sadayappan, P. "Task allocation by simulated annealing," *Proc. International Conference on Supercomputing.* Boston, MA, May 1988, vol. III, *Hardware & Software*, pp. 475-497.
- [24] Ramanujam, J., and Sadayappan, P. "Optimization by Neural Networks," *IEEE Int. Conference on Neural Nets*, Vol.II, pp. 325-332, July 1988.
- [25] Sadayappan, P., and Erçal, F. "Nearest-neighbour mapping of finite element graphs onto processor meshes," *IEEE Trans. Comput.* vol. 36, no. 12, pp. 1408-1424, 1987.
- [26] Sadayappan, P., Erçal, F., and Ramanujam, J. "Cluster partitioning approaches to mapping parallel programs onto a hypercube," *Parallel Computing.* vol. 13, pp. 1-16, 1990.
- [27] Schweikert, D. G., and Kernighan, B. W. "A proper model for the partitioning of electrical circuits," in *Proc. 9th Design Automat. Workshop*, pp. 57-62, 1979.
- [28] Seitz, C. L. "The Cosmic Cube," *Com. of the ACM*, vol. 28, pp. 22-23, 1985.
- [29] Shield, J. "Partitioning concurrent VLSI simulation programs onto a multiprocessor by simulated annealing," *IEEE Proc. Part G*, vol. 134, no. 1, pp. 24-28, 1987.
- [30] Szu, H. "Fast TSP Algorithm Based On Binary Neuron Output and Analog Neuron Input Using The Zero-Diagonal Interconnect Matrix And Necessary And Sufficient Constraints Of The Permutation Matrix," *IEEE Int. Conference on Neural Nets*, Vol.II, pp. 259-266, July 1988.

- [31] Tank, D. W., and Hopfield, J. J. "Simple 'Neural' optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circ. Syst.*, Vol.cas-33, no.5, May 1986.
- [32] Toomarian, N. "A Concurrent Neural Network Algorithm for the Traveling Salesman Problem," *Third Conference on Hypercube Concurrent Computers and Applications*, Pasadena.
- [33] Van den Bout, D. E., and Miller, T. K. "A Traveling Salesman Objective Function That Works," *IEEE Int. Conf. Neural Nets*, vol. 2, pp. 299-303, 1988.
- [34] Van den Bout, D. E. and Miller, T. K. "Improving the performance of the Hopfield-Tank neural network through normalization and annealing," *Biolog. Cybern.*, vol. 62, pp. 129-139, 1989.
- [35] Van den Bout, D. E., and Miller, T. K. "Graph partitioning using annealed neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 192-203, 1990.
- [36] Wilson, G. V., and Pawley, G. S. "On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank," *Biolog. Cybern.*, vol. 58, pp. 63-70, 1988.
- [37] Yih, J. S., and Mazumder, P. "A neural network design for circuit partitioning," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1265-1271, 1990.