

MAPPING
AND
FPGA GLOBAL ROUTING
USING
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
Ismail Karitaoglu
September, 1994

Thesis
T.K
7895
.G36
K37
1994

MAPPING
AND
FPGA GLOBAL ROUTING
USING
MEAN FIELD ANNEALING

A THESIS

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

By

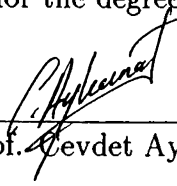
İsmail Harıtaoğlu
September, 1994

İsmail HARİTAOĞLU
tarafından bağışlanmıştır

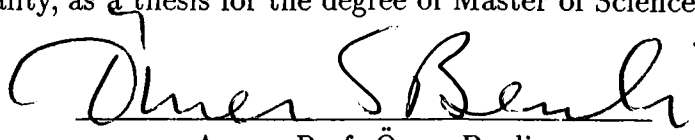
TK
7895
.G36
H37
1994

B026792

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. Cevdet Aykanat (Advisor)

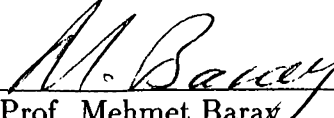
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Assoc. Prof. Ömer Benli

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. Mustafa Pınar

Approved for the Institute of Engineering and Science:


Prof. Mehmet Baray
Director of the Institute

ABSTRACT

MAPPING
AND
FPGA GLOBAL ROUTING
USING
MEAN FIELD ANNEALING

İsmail Harिताoğlu
M.S. in Computer Engineering and Information Science
Advisor: Asst. Prof. Cevdet Aykanat
September, 1994

Mean Field Annealing algorithm which was proposed for solving combinatorial optimization problems combines the properties of neural networks and Simulated Annealing. In this thesis, MFA is formulated for mapping problem in parallel processing and global routing problem in physical design automation of Field Programmable Gate Array (FPGAs). A new Mean Field Annealing (MFA) formulation is proposed for the mapping problem for mesh-connected and hypercube architectures. The proposed MFA heuristic exploits the conventional routing scheme used in mesh and hypercube interconnection topologies to introduce an efficient encoding scheme. An efficient implementation scheme which decreases the complexity of the proposed algorithm by asymptotical factors is also developed. Experimental results also show that the proposed MFA heuristic approaches the speed performance of the fast Kernighan-Lin heuristic while approaching the solution quality of the powerful simulated annealing heuristic. Also, we propose an order-independent global routing algorithm for SRAM type FPGAs based on Mean Field Annealing. The performance of the proposed global routing algorithm is evaluated in comparison with LocusRoute global router on *ACM/SIGDA Design Automation* benchmarks. Experimental results indicate that the proposed MFA heuristic performs better than the LocusRoute.

Keywords: Mapping, Global Routing, Field Programmable Gate Arrays, Mean Field Annealing

ÖZET

ORTA ALAN TAVLAMA METODU KULLANILARAK EŞLEME VE FPGA'LERDEKİ KABA ROTALAMA PROBLEMLERİNİN ÇÖZÜMÜ

İsmail Haritaoğlu

Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans

Danışman: Yrd. Doç. Dr. Cevdet Aykanat

Eylül, 1994

Birleşimsel eniyileme problemlerini çözmek için önerilen Ortak Alan Tavlama (Mean Field Annealing) algoritması, sinir ağları ve benzetimsel tavlama (Simulated Annealing) yöntemlerinin özelliklerini taşır. Bu çalışmada, Ortak Alan Tavlama algoritması Alan Programlamalı Kapı Devrelerinin (Field Programmable Gate Arrays) kaba rotalama problemine (Global Routing) ve paralel programlamadaki eşleme (Mapping) problemlerine uyarlanmıştır. Tezin ilk bölümünde Ortak Alan Tavlama algoritması Alan Programlamalı Kapı Devrelerinin (Field Programmable Gate Arrays) kaba rotalama problemi-nin çözümünde kullanılmıştır. Önerilen algoritmalarının başarımları Locus-Route kaba rotalama algoritması ile kıyaslanarak değerlendirilmiştir. Deneyler algoritmaları karşılaştırmak için kullanılan standart devreler (Benchmarks) üzerinde yapılmıştır. Elde edilen sonuçlar Ortak Alan Tavlama algoritmasının kaba rotalama problemini çözmek için iyi bir alternatif algoritma olarak kullanılabilirliğini göstermektedir. Tezin ikinci bölümünde Mesh ve Hiperküp tipindeki paralel bilgisayarlarındaki eşleme problemi için daha önce önerilen algoritmalarından daha hızlı olan bir algoritma geliştirilmiş ve bu önerilen algoritmanın başarımları Kernighan-Lin, Simulated Annealing ve daha önce önerilen ortak alan tavlama metotları ile kıyaslanarak değerlendirilmiştir.

Anahtar Sözcükler: Orta Alan tavlama algoritması, Eşleme problemi, Kaba rotalama algoritmaları, Alan programlamalı kapı devreleri

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor Dr. Cevdet Aykanat for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis. I would like to thank Dr. Ömer Benli for reading and commenting on the thesis. I would also like to thank Dr. Mustafa Pınar for reading and commenting on the thesis. I owe special thanks to Dr. Mehmet Baray for providing a pleasant environment for study. I am grateful to my family and my friends for their infinite moral support and help.

Bu alıřmamı,
herřeyimi borlu olduėum *anneme, babama,*
ve
Esin'e
adıyorum.

Contents

1	INTRODUCTION	1
2	MEAN FIELD ANNEALING	4
2.1	Mean Field Annealing	4
2.1.1	Ising Model	5
2.1.2	Potts Model	6
2.1.3	MFA Algorithm	8
3	FPGAs & GLOBAL ROUTING	9
3.1	Introduction to Field Programmable Gate Arrays	9
3.1.1	Logic Blocks	10
3.1.2	Programming Technologies	10
3.1.3	Routing Architectures	11
3.2	Physical Design Automation of FPGAs	15
3.2.1	Partitioning	15
3.2.2	Placement	15
3.2.3	Routing	15
3.3	Global Routing Problem in Design Automation of FPGAs	16

3.4	Model of FPGA for Global Routing	17
4	MFA SOLUTION FOR GLOBAL ROUTING IN FPGA	22
4.1	MFA Formulation of Global Routing	22
4.2	Implementation	25
4.3	Experimental Results	27
5	THE MAPPING PROBLEM	33
5.1	The Mapping Problem	33
5.2	The Model of Mapping Problem	35
6	MFA SOLUTION FOR MAPPING	39
6.1	General MFA Formulation for Mapping Problem	39
6.2	Interconnection-Topology Specific MFA Formulation for Mapping	42
6.2.1	MFA formulation for Mesh-Connected Architectures . . .	42
6.2.2	MFA Formulation For Hypercube Architecture	51
6.3	Performance Evaluation	56
6.4	Experimental Results	59
7	CONCLUSION	69

List of Figures

2.1	Mean Field Annealing Algorithm	8
3.1	The Architecture of General FPGA	11
3.2	Example of flexibilities of FPGA (a) flexibility of switch block (b) flexibility of connection block	12
3.3	The Architecture of Xilinx 3000 FPGA	13
3.4	The Architecture of Actel FPGA	14
3.5	General approach to FPGA routing a) Global routing b) De- tailed routing	16
3.6	Sample two bends routes	17
3.7	The FPGA model used for Global Routing	18
3.8	(a) The routing area of the two-pin net and its subnets, (b) The possible routes for each subnets	19
3.9	The Cost Graph for FPGA model	20
4.1	Channel density distribution obtained by MFA for the circuit <i>C1355</i>	32
4.2	Channel density distribution obtained by LocusRoute for the circuit <i>C1355</i>	32
4.3	SEGA detailed router results of the circuit <i>C1355</i> for the global routing solutions obtained by (a) MFA (b) LocusRoute	32

5.1	An example of mapping problem	38
6.1	The proposed efficient MFA algorithm for the mapping problem for mesh-connected Architectures.	48
6.2	Three different ways for dividing 3-dimensional hypercube to 2 2-dimensional subcubes	52
6.3	The Mean field value calculation of given spin i of subcube H^m .	56

List of Tables

4.1	MCNC benchmark circuits used in experiments	27
4.2	The Global Router results	28
4.3	The SEGA detailed routing results in area optimization mode .	29
4.4	The SEGA detailed routing results in speed optimization mode .	30
4.5	Minimum Channel Width for 100% routing	31
6.1	Total communication costs averages normalized with respect to mesh-specific MFA of the solution found by SA,KL,general MFA and mesh-specific MFA for randomly generated mapping prob- lem instances for various mesh size	59
6.2	Percent computational load imbalance averages of the solution found by SA,KL,general MFA and mesh-specific MFA for ran- domly generated mapping problem instances for various mesh size	60
6.3	Execution time averages of the solution found by SA,KL,general MFA and mesh-specific MFA for randomly generated mapping problem instances for various mesh size	60
6.4	Average performance measures of the solution found by SA, KL, general MFA and mesh-specific MFA for randomly generated mapping problem instances	61
6.5	The Benchmark Sparse Matrix data used in experiments	62

6.6	Total communication cost averages, normalized with respect to mesh-specific MFA, of the solution found by SA, KL, general MFA and mesh-specific MFA for some benchmark mapping problem instances for various mesh size	63
6.7	Load Imbalanced averages, of the solution found by SA, KL, general MFA and mesh-specific MFA for some benchmark mapping problem instances for various mesh size	64
6.8	Total execution time, normalized with respect to mesh-specific MFA, of the solution found by SA, KL, general MFA and mesh-specific MFA for some benchmark mapping problem instances for various mesh size	65
6.9	Average performance measures of the solutions found by SA, KL, general MFA and mesh-specific MFA for mapping problem instances.	66
6.10	Total communication costs averages normalized with respect to hypercube-specific MFA of the solution found by SA, KL, general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercube size	66
6.11	Percent computational load imbalance averages of the solution found by SA, KL, general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercube size	67
6.12	Execution time averages of the solution found by SA, KL, general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercube size	67

Chapter 1

INTRODUCTION

A common property of both domain mapping problem in parallel processing and global routing in VLSI is that both problems are combinatorial optimization problems. As many problems in VLSI, parallel processing and other areas, these algorithms involve a finite set of configuration from solutions satisfying a number of rigid requirement are selected. The objective of combinatorial optimization algorithm is to find a solution of the optimum cost provided that a cost can be assigned to each solution. Many combinatorial optimizations problems are hard in the sense that they are NP-hard problems. There are no known deterministic polynomial time algorithms to find the optimal solution to any of those hard problems. The algorithms using the complete enumeration techniques are usually exponential in the size of problem, therefore they require a great amount of time to find the optimal solution. As a result, heuristics that run in a low order polynomial time have been employed to obtain good solutions to these hard problems. Disadvantage of heuristics is that they may get stuck in local minima.

A powerful method for solving combinatorial optimization problem used in previous research is called *Simulated Annealing*. 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. Simulated Annealing is a general method that guarantees to find the optimal solution if time is not limited. But time needed for Simulated Annealing is too much and exact solution of NP-hard problems are still intractable. Properties of Simulated Annealing are 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. An important property of Simulated Annealing is the ability to escape from local minima if sufficient time is given. Simulated Annealing has been applied to various NP-hard optimization problem and for most problem it gives good results.

The subjects of this thesis is a recent algorithm, called Mean Field Annealing (MFA) was originally proposed for solving the traveling salesperson problem. MFA is general strategy and can be applied to various problem with suitable formulations. Work on MFA showed that, it can be successfully applied to combinatorial optimization problems. Mean Field Annealing (MFA) merges collective computation and annealing properties of Hopfield Neural Networks (HNN) and Simulated Annealing (SA), respectively, to obtain a general algorithm for solving combinatorial optimization problems. MFA can be used for solving a combinatorial optimization problem by choosing a representation scheme in which the final states of the *spins* can be decoded as a solution to the target problem. Then, an energy function is constructed whose global minimum value corresponds to the *best solution* of the problem to be solved. MFA is expected to compute the *best solution* to the target problem, starting from a randomly chosen initial state, by minimizing this energy function. In this thesis, MFA is formulated for the mapping problem in parallel processing and global routing problem in design automation of Field Programmable Gate Arrays.

The first combinatorial optimization problem, that is solved by MFA in this thesis, is global routing problem in design automation of field programmable gate arrays. This study investigates the routing problem in Static RAM Field Programmable Gate Arrays (FPGA's) implementing the non-segmented (Xilinx based) network [27]. As the routing in FPGA's is a very complex combinatorial optimization problem, routing process can be carried out in two phases : *global routing* followed by *detailed routing* [11]. Global routing determines the course of wires through sequences of channel segments. Detail routing determines the wire segment allocation for the channel segment routes found in the first phase which enable feasible switch box interconnection configurations [25, 14]. Global routing in FPGA can be done by using global routing algorithm proposed for standard cells [25]. *LocusRoute* global router is one of this type of router used for global routing in FPGA's [24] which divides the multi pin net's into two-pin net's and considers only minimum distance routes for these two-pin nets. The objective in the *LocusRoute* is to distribute the connections among channels so that channel densities are balanced. In this

thesis, we propose a new approach the solution of global routing problem in FPGA's by using Mean Field Annealing technique.

Second problem that is solved by MFA is the *Mapping* problem [4, 8, 29]. The mapping problem arises as parallel programs are developed for distributed memory architectures. Various classes of problems can be decomposed into a set of interacting sequential subproblems (tasks) which can be executed in parallel. In these classes of problems, the interaction patterns among the tasks is static. In a distributed-memory architecture, a pair of processors communicate with each other over a shortest path of links connecting them. Hence, communication between each pair of processors can be associated with relative unit communication cost. Unit communication cost between a pair of processors can be assumed to be linearly proportional to the shortest path distance between those two processors. The objective in mapping subproblems to processors of multicomputers is the minimization of the expected execution time of the parallel program on the target architecture. Thus, the mapping problem can be modeled as an optimization problem by associating the following quality measures with a good mapping : (i) interprocessor communication overhead should be minimized, (ii) computational load should be uniformly distributed among processors in order to minimize processor idle time. The mapping problem has been solved by using Simulated Annealing, Kernighan-Lin type heuristic before. Also the MFA has been formulated in [6, 5]. But this formulation was a general formulation for any type of multicomputer whose intercommunication topologies are known. In this thesis we propose an efficient MFA formulation for topology-specific mapping for 2D-mesh and hypercube. For each interconnection topology, the efficient MFA formulation is given instead of using one general formulation as in [6].

In Chapter 2 the theory of the Mean Field Annealing heuristic and its encoding models are explained. The Field Programming Gate arrays, its design automation and Global Routing problem are introduced in Chapter 3. Also the FPGA model for global routing problem are proposed in this chapter. Chapter 4 gives the MFA formulation of global routing problem in FPGAs design automation. The mapping problem are introduced in Chapter 5. Chapter 6 presents general MFA formulation the topology-specific MFA formulation for Domain Mapping problem. Finally, conclusion of thesis are stated in Chapter 7.

Chapter 2

MEAN FIELD ANNEALING

In this chapter the Mean Field Annealing (MFA) heuristic is introduced and its models are given.

2.1 Mean Field Annealing

Mean Field Annealing (MFA) merges collective computation and annealing properties of Hopfield Neural Networks (HNN) and Simulated Annealing (SA), respectively, to obtain a general algorithm for solving combinatorial optimization problems. HNN is used for solving various optimization problems and reasonable results are 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. Hence, the algorithm does not have a good scaling property, which is a very important performance criterion for heuristic optimization algorithms. MFA is proposed as a successful alternative to HNN. In the MFA algorithm, problem representation is identical to HNN, but iterative scheme used to relax the system is different. MFA can be used for solving a combinatorial optimization problem by choosing a representation scheme in which the final states of the *spins* can be decoded as a solution to the target problem. Then, an energy function is constructed whose global minimum value corresponds to the *best solution* of the problem to be solved. MFA is expected to compute the *best solution* to the target problem, starting from a randomly chosen initial state, by minimizing this energy function. Steps of formulating MFA technique for a combinatorial optimization problem can be summarized as follows :

- Choose a representation scheme which encodes the configuration space of the target problem using spins. In order to get a good performance, number of possible configurations in the problem domain and the spin domain must be equal, i.e., there must be a one-to-one mapping between the configurations of spins and the problem.
- Formulate the cost function of the problem in terms of spins, i.e., derive the energy function of the system. Global minimum of the energy function should correspond to the global minimum of the cost function.
- Derive the mean field theory equations using this energy function, i.e., derive equations for updating expected values of spins.
- Minimize the complexity of update operations in order to get an efficient algorithm.
- Select the energy function and the cooling schedule parameters.

The MFA algorithm is derived by analogy to *Ising* and *Potts model* which are used to estimate the state of a system of particles, called spins, in thermal equilibrium.

2.1.1 Ising Model

In Ising model spins can be in one of two states represented by 0 and 1. In the Ising model, the energy of a system with S spins has the following form:

$$H(\mathbf{s}) = \frac{1}{2} \sum_{k=1}^S \sum_{l \neq k}^S \beta_{kl} s_k s_l + \sum_{k=1}^S h_k s_k \quad (2.1)$$

Here, β_{kl} indicates the level of interaction between spins k and l , and $s_k \in \{0, 1\}$ is the value of spin k . 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

$$\langle s_k \rangle = \frac{1}{1 + e^{-\phi_k/T}} \quad (2.2)$$

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

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

The complexity of computing ϕ_k using Eq.2.3 is exponential. However, for large number of spins, *mean field approximation* can be used to compute the energy average as

$$\langle H(\mathbf{s}) \rangle = \frac{1}{2} \sum_{k=1}^S \sum_{l \neq k} \beta_{kl} \langle s_k \rangle \langle s_l \rangle + \sum_{k=1}^S h_k \langle s_k \rangle \quad (2.4)$$

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

$$\phi_k = \langle H(\mathbf{s}) \rangle|_{s_k=0} - \langle H(\mathbf{s}) \rangle|_{s_k=1} = -\frac{\partial \langle H(\mathbf{s}) \rangle}{\partial \langle s_i \rangle} = -\left(\sum_{l \neq k} \beta_{kl} \langle s_l \rangle + h_k \right) \quad (2.5)$$

2.1.2 Potts Model

In the Potts model, spins can be in one of the K states. In K state Potts model of S spins, the states of spins are represented using S K -dimensional vectors $\mathbf{S}_i = [s_{i1}, \dots, s_{ik}, \dots, s_{iK}]^t$, $1 \leq i \leq S$, where “ t ” denotes the vector transpose operation.

The spin vector \mathbf{S}_i is allowed to be equal to one of the principal unit vectors $\mathbf{e}_1, \dots, \mathbf{e}_k, \dots, \mathbf{e}_K$, and can not take any other value. Principal unit vector \mathbf{e}_k is defined to be a vector which has all its components equal to 0 except its k 'th component which is equal to 1. Spin \mathbf{S}_i is said to be in state k if it is equal to \mathbf{e}_k . Hence, a K state Potts spin \mathbf{S}_i is composed of K two state variables $s_{i1}, \dots, s_{ik}, \dots, s_{iK}$, where $s_{ik} \in \{0, 1\}$, with the following constraint

$$\sum_{k=1}^K s_{ik} = 1, \quad 1 \leq i \leq S \quad (2.6)$$

In the Potts model, the energy of a system with S K -state Potts spin has the following form:

$$E(\mathbf{s}) = \frac{1}{2} \sum_{i=1}^S \sum_{j \neq i} \beta_{ij} \mathbf{S}_i \mathbf{S}_j + \sum_{i=1}^S e_i \mathbf{S}_i \quad (2.7)$$

Here, β_{ij} indicate the level of interaction between spins i and j , and interaction between Potts spins $\mathbf{S}_i \mathbf{S}_j$ is formulated as $1/2 \sum_{k=1}^K \sum_{l=1}^K s_{ik} s_{jl}$. Therefore we

can formulate the energy of the system as

$$E(s) = \frac{1}{2} \sum_{i=1}^S \sum_{j \neq i} \beta_{ij} 1/2 \sum_{k=1}^K \sum_{l=1}^K s_{ik} s_{jl} + \sum_{i=1}^S e_i \sum_{k=1}^K s_{ik} \quad (2.8)$$

Here, $s_{ik} \in 0, 1$ is the value of k th state of the Potts spin i . At thermal equilibrium, spin average $\langle s_{ik} \rangle$ of spin i can be calculated using Boltzmann distribution as follows

$$\langle s_{ik} \rangle = \frac{e^{\phi_{ik}/T}}{\sum_{l=1}^K e^{\phi_{il}/T}} \quad (2.9)$$

Here, $\langle s_{ik} \rangle \in [0, 1]$. Note that s_{ik} can be 0 or 1 but $\langle s_{ik} \rangle$ can be any real value between 0 and 1. ϕ_{ik} represents the *mean field* effecting on state k of spin i . The mean field value for Potts spin i can be formulated as

$$\phi_{ik} = \langle E(s) \rangle |_{s_i=0} - \langle H(s) \rangle |_{s_i=e_k} \quad (2.10)$$

$$= -\frac{\partial \langle H(s) \rangle}{\partial \langle s_{ik} \rangle} = -\left(\sum_{j \neq i} \beta_{ij} \sum_{l=1}^K s_{jl} + h_i \right) \quad (2.11)$$

At each temperature, starting with initial spin averages, the mean field effecting on a randomly selected spin is found using Eqs. (2.5) and (2.10). Then, spin average is updated using Eq. (2.2) and Eq. (2.9). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. 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, 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 [9]. 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 [8, 9]. 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.

-
1. Get the Initial temperature T_0 , and set $T = T_0$
 2. Initialize spin averages
 - Ising spin : $\{\langle u_1 \rangle, \langle u_2 \rangle, \dots\}$
 - Potts spins : $\{\langle S_1 \rangle, \langle S_2 \rangle, \dots\}$
 3. WHILE temperature T is in the cooling range DO
 4. WHILE system is not stabilized for the current temperature DO
 - Select a spin i at random
 - 4.1 Compute mean field affecting on spin i
 - Ising spin : compute $\phi_i = E(\mathbf{U})|_{u_i=0} - E(\mathbf{U})|_{u_i=1}$
 - Potts spins : compute $\phi_i = [\phi_{i1}, \phi_{i2}, \dots, \phi_{iK}]^t$ such that

$$\phi_{ik} = E(\mathbf{S})|_{s_i=0} - E(\mathbf{S})|_{s_i=e_k} \text{ for } k = 1, 2, \dots, K$$
 - 4.2 Update the average value of spin i
 - Ising spin: $\langle u_i \rangle = e^{\phi_i/T} / (1 + e^{\phi_i/T})$
 - Potts spin : $\langle s_{ik} \rangle = e^{\phi_{ik}/T} / \sum_{l=1}^K e^{\phi_{il}/T}$ for $k = 1, 2, \dots, K$
 5. Update T according to the cooling schedule
-

Figure 2.1. Mean Field Annealing Algorithm

2.1.3 MFA Algorithm

The Mean Field Annealing algorithm are summarized in Figure 2.1.2. Beginning of the algorithm, the initial temperature are initialized and the current temperature is set to that initial value (step 1). After that Ising and Potts spins are initialized (step 2). Then, the annealing property of MFA are begin. In cooling schedule, the system tries to reach a stable state for each temperature until most of spins converges a stable state. For each temperature, while the system is not in stable state, a spin is selected randomly (step 4.1), and mean field values of spins are calculated (step 4.2) in order to update the spin values (step 4.3). When the system reaches the stable state, the temperature decreased by cooling schedule (step 5). At the end of algorithm, when most of spins converge, spins are decoded for a solution of target problem.

Chapter 3

FPGAs & GLOBAL ROUTING

This chapter introduces the Field Programmable Gate Arrays and its physical design automation steps briefly. Routing architectures of FPGA's are mentioned in this chapter and global routing problem and its previous solutions are given at the end of this chapter. Also the global routing problem in FPGAs is modeled in this chapter.

3.1 Introduction to Field Programmable Gate Arrays

Field Programmable gate arrays (FPGAs) are new electrically programmable integrated circuits that provide high integration and rapid turnaround time. In VLSI design automation, the fabrication time is important problem. In order to reduce time to fabricate interconnects, programmable devices have been introduced. FPGA is very popular programmable devices used in ASIC design market.

FPGA can reduce manufacturing turnaround time and cost. In its simplest form, an FPGA consists of an array of programmable logic blocks and routing network to interconnect the logic blocks. The programmable logic blocks can be programmed by the user to implement a small logic function. An important property of FPGA is re-programmability by using electrically programmable switches. Commercial FPGA's differ in the type of programming technology used, in architecture of logic blocks and their routing architectures. An FPGA logic blocks can be as simple as transistor or as complex as a microprocessor.

It is typically capable of implementing many different combinational and sequential logic functions. FPGA's logic blocks can be classified as transistors pairs, basic small gates (such as two-input NAND's), multiplexes and Look-up tables.

3.1.1 Logic Blocks

FPGAs logic blocks differ greatly in their size and implementation capability. The two transistor logic block can only implement an inverter but is very small in size, while look-up table logic blocks used in Xilinx FPGAs can implement any five-input logic function but they are significantly larger. Logic blocks can be classified in terms of granularity. Granularity can be defined in various ways, for example, as the number of boolean function that the logic block can implement, the number of equivalent two input NAND gates, total number of transistors, number of inputs and outputs. But generally, the commercial logic blocks can be classified into two categories: *fine-grain* and *coarse-grain*. Main advantage of using fine grain logic blocks is that the use-able blocks are fully utilized. However the main disadvantage of fine-grain blocks is that they require a relatively large number of wire segments and programmable switches.

3.1.2 Programming Technologies

An FPGA is programmed using electrically programmable switches. According the properties of these programmable switches such as, on-resistance and capacitance, programming technologies can be classified into three main types. These three types are SRAM , antifuse and EPROM programming technologies.

The SRAM programming technologies uses static RAM cells to control the gates and multiplexes. In SRAM, the switch is a pass transistor controlled by the state of a SRAM bit. Therefore, SRAM is volatile. Hence The FPGA must be loaded and configured at the time of chip power-up, it requires external permanent memory to provide the programming bits such as PROM or EPROM. A major disadvantage of SRAM programming technology is its large area (it takes at least five transistors to implement an SRAM cell). However, SRAM programming technology has fast re-programmability as an advantage of it.

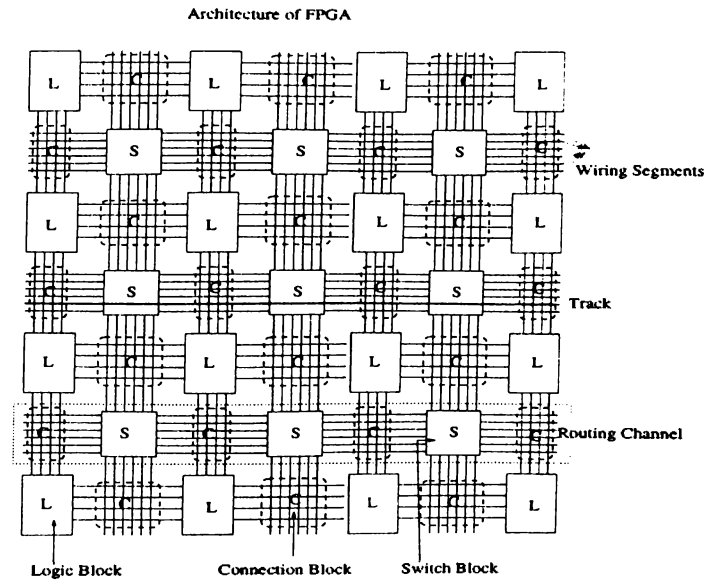


Figure 3.1. The Architecture of General FPGA

An antifuse is a two terminal device with an unprogrammed state presenting a very high resistance between its terminals. When a high voltage is applied across its terminals, the antifuse will blow and create low resistance link. This link is permanent. Programming an antifuse requires extra circuitry to deliver the high programming voltage and a high current. A major advantage of the antifuse is its small size. This advantage is reduced by the large size of the necessary programming transistors.

The floating gate programming technology uses technology found in ultra-violet erasable EPROM and electrically erasable EEPROM. Major advantage of EPROM technology is its fast reprogrammability. Also it does not require extra permanent memory to program the chip on power-up. However this technology increase the number of processing steps and high resistance transistors.

3.1.3 Routing Architectures

The routing architecture of an FPGA is the manner in which the programmable switches and wiring segments are positioned to allow the programming inter-connection of the logic. Figure 3.1 illustrates a typical routing architecture model. Before giving some commercial FPGA routing architecture, giving some definition is helpful for understand routing problem in FPGA. A *wire*

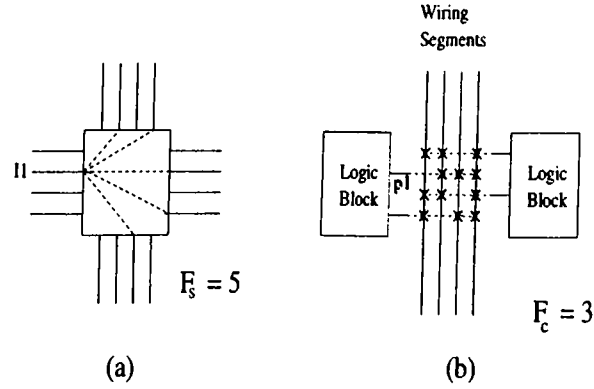


Figure 3.2. Example of flexibilities of FPGA (a) flexibility of switch block
(b) flexibility of connection block

segment is a wire unbroken by programmable switches. One or more switches may attach to the wire segment. Each end of wire segment has a switch attached.

A *track* is sequence of one or more wire segments in a line.

A *routing channel* is group of parallel tracks as in Figure 3.1.

As shown in Figure 3.1, the model contains two basic structures: Connection blocks and switch blocks. A connection block provides connectivity from the input and output of logic blocks to the wire segments in the channels. A switch block provides connectivity between the horizontal as well as the vertical wire segments.

As in Figure 3.2, The general routing structure of FPGA has two important interconnection block. These are connection blocks which are used to make connections between logic block pin and routing segments, and switch blocks where connections are switched at the intersection of horizontal and vertical channels. The number of switching in connection and switch blocks is important for good routability. Large number of switching increase the routability but it causes poor performance and large delay and also large area.

The number and distribution of switches used in interconnection called *flexibility* of an FPGA. Flexibility of switch blocks (F_s) and flexibility of connection block (F_c) can be defined as the number of choices offered to each wire enter a switching block or a connection block, respectively. The *flexibility of switch block* F_s is defined to be total number of possible connection offered

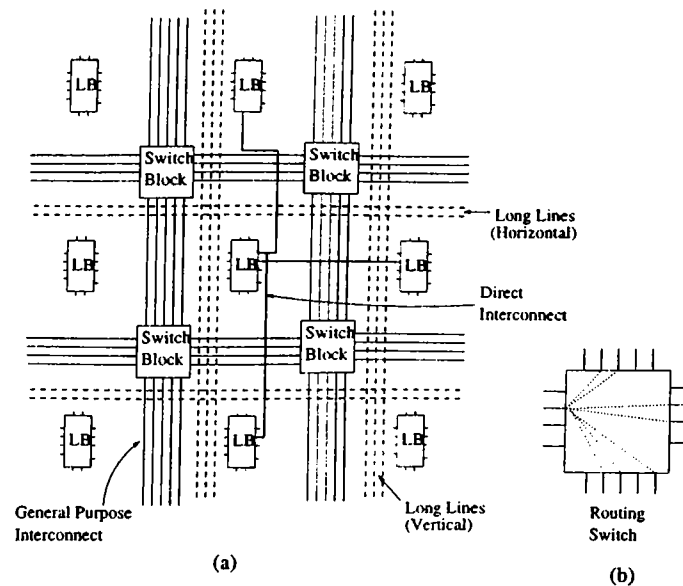


Figure 3.3. The Architecture of Xilinx 3000 FPGA

to each wire segment. The *flexibility of connection block* F_c is defined as the number of wires that each logical pin of logic block can connect. Next section describes the important routing architecture of commercial FPGA's such as Xilinx and Actel.

The Xilinx Routing Architecture

Figure 3.3 illustrates the routing architecture used in the Xilinx 3000 series FPGA. Connections are made from the logic block into the channel through a connection block. Since each connection site is large because of the SRAM programming technology, the Xilinx 3000 connection blocks connects each pin to only two or three out of five tracks passing by a block. On all four sides of the logic block there are connection blocks that connect a total of 11 different logic block pins to the wire segments. Once the logic pin is connected via the connections block makes connections between segments in intersecting horizontal and vertical channels. Each wire segment can connect to five or six out of a possible 15 wire segments on the opposites sides. There are four types of wire segments provided in the Xilinx 3000 architecture:

- General-purpose interconnect consisting of wire segments that pass through switches in the switch block.

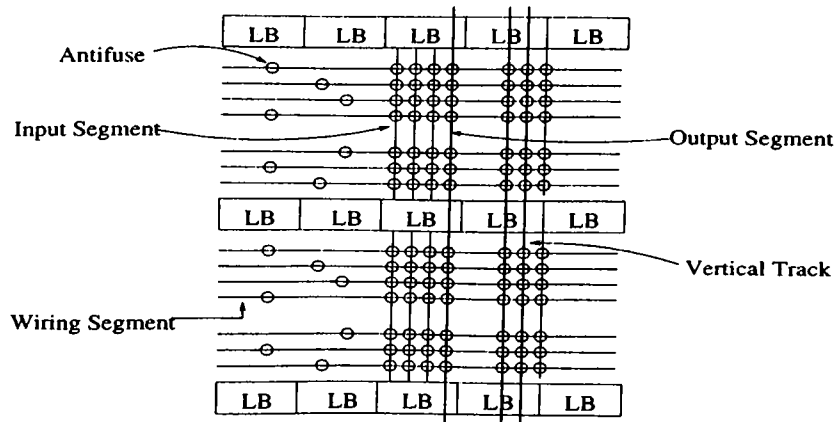


Figure 3.4. The Architecture of Actel FPGA

- Direct interconnect consisting of wire segments that connect each logic block output directly to four nearest neighbors.
- Long lines, which span the length or width of the chip, providing high-fanout uniform delay connections.
- Clock line, which is a single net that spans the entire chip and is driven by a high-drive buffer.

The Actel Routing Architecture

The Actel routing architecture has a asymmetric architecture because there are more general purpose tracks in horizontal direction than vertical direction. The connection block of the Actel routing architecture is shown in Figure 3.4. The connectivity of ACTEL FPGAs is different in input and output pins. For input pins, each pin can connect to all of the tracks in the channel that are on the same side as the pin. The output pins extend across two channels above the logic block and two channels below it. Output pins can connect to every track in all four channels that is crosses. There is no separable switch block in the Actel architecture. Instead, the switching is distributed throughout the horizontal channels. All vertical tracks can make a connection with every incident horizontal tracks. Each horizontal channel consists of 22 routing tracks, and each track is broken up into segments of different lengths. There are three type of vertical segments: input segments, output segments and freeways that either travel the entire height of chip, or some significant portion of it. This allows signal to travel longer vertical distance than permitted by output segments.

3.2 Physical Design Automation of FPGAs

The physical design automation of FPGAs involves mainly three steps which include partitioning, placement and routing.

3.2.1 Partitioning

Partitioning is the separation of the logic into Logic blocks. Partitioning has both a logical and physical component. The connections within a logic blocks are constrained by the limited routing architecture and limited number of blocks outputs. However, the quality of the resulting partitioning depends on how well the placement can be done. The logical component has been investigated in the context of technology mapping in logic optimization.

3.2.2 Placement

Placement starts with logic blocks and input-output blocks in partitioned netlist and decides which corresponding blocks on the chip should contain them. The FPGA placement problem is very similar to traditional standard cell and gate array placement problems. Many of existing algorithm placement algorithms are applicable, such as simulated annealing, force directed relaxation and min-cut.

3.2.3 Routing

After placement of all circuit, each pin of any multipoint net have to be connected. There are several routing algorithms for different kind of FPGA architectures and routing problem in FPGA's is very complex as in standard cells and gate arrays designs. Because of simplicity, the routing problem can be divided into two step as in traditional routing problem: global routing and detailed routing.

Global routing in FPGA's can be done by using a global router for standard cell design. In general such a global router divides the multipoint nets into two terminal nets and routes them with minimum distance path. While doing so it

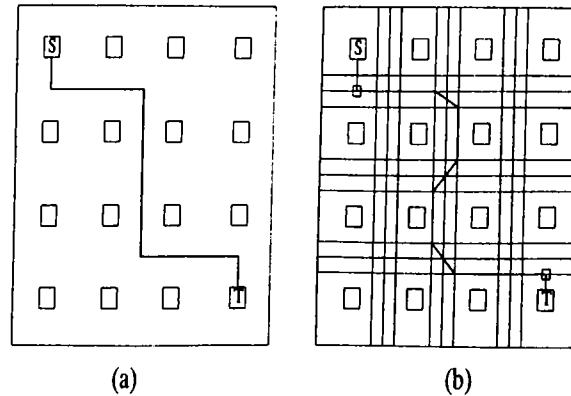


Figure 3.5. General approach to FPGA routing a) Global routing b) Detailed routing

also tries to balance the density of channels. The global route defines a coarse route for each connection by assigning it a sequence of channel segments. After the paths are defined in terms of channel between two-pin connection detailed router chose specific wiring segments to implement the channel segment assigned during global routing.

3.3 Global Routing Problem in Design Automation of FPGAs

A global router chooses channels for each net and leaves the task of allocating specific wiring segments and switches to detailed router. The global routing in FPGA's decides for each net to determine which pins are actually to be connected. The objective of global router is to minimize the sum of the channel densities of all channels. As in many studies, the routing problem in FPGA is solved by directly allocating the segments and ignore the global routing phase. There are unique global router for FPGA: PGAroute. This global router similar the global router for standard cells and use the LocusRoute global routing algorithm.

In the LocusRoute algorithm, the following three steps are executed for each multi pin nets.

1) Net's Division: Each multi-pin net is divided into a set of two-pin connections using a minimum spanning tree algorithm.

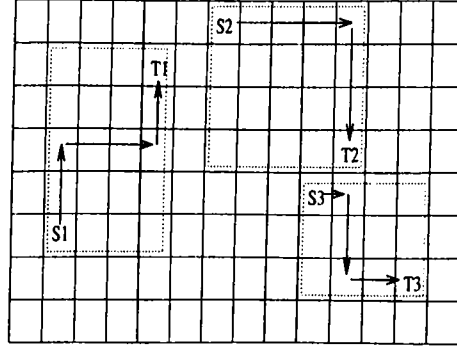


Figure 3.6. Sample two bends routes

2) Route Generation and Evaluation: In this steps, the possible paths between each pin of two-pin nets are considered and evaluate this paths in terms of cost value and chose the lowest cost value path.

The method of choosing routes is based on paths that have two or less bends. LocusRoute evaluates a subnet of all two bend routes between the two physical pins and chose the one with the lowest cost. The cost function is defined in terms of the channel densities. Each wire segments and switch blocks are represented as elements of an array which is called as cost array. Each element of cost array $H_{i,j}$ contains the number of routes that pass through the wire segment of (i,j) . The cost of path(P) is calculated as

$$Cost(P) = \sum_p H_{i,j} \quad (3.1)$$

3)Reconstruction: This step joins all two-pin connections back together, performs assigns unique numbers to distinct segments of some nets in each channel.

Locus routes uses the iterative technique, that after the first time all nets are routed, each is sequentially ripped up and rerouted. Iterations reduces the order dependency and also it improves the routing quality.

3.4 Model of FPGA for Global Routing

The form of commercial FPGA consists of a two dimensional regular array of programmable logic blocks (LB's), a programmable routing network and

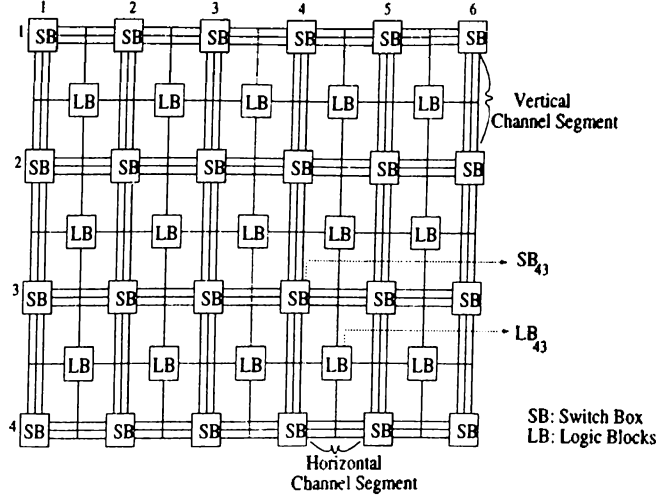


Figure 3.7. The FPGA model used for Global Routing

switch boxes (SB's) [3, 1, 2]. Logic blocks are used to provide the functionality of a circuit. Routing network makes connections between LB's and input/output pads. Routing network of FPGA consists of wiring segments and connection blocks. Wiring segments have three type of routing resources in the commercial SRAM based FPGA [1]: channel segments, long lines and direct-interconnections. A horizontal (vertical) channel segment consists of a number of parallel wire segments connecting two successive SB's in a horizontal (vertical) channel. The SB's allow programmed interconnection between these channel segments. Direct-interconnection provides the connections between neighbor LB's. Long lines cross the routing area of FPGA vertically and horizontally. Connection blocks provide the connectivity from the input/output pins of LB's to the wiring segments of the respective channel segments. Each pin can be connected to a limited number of wiring segments in a channel and this is called as flexibility of connection block [16]. In this work, it is assumed that each LB pin can be connected to all wiring segments in the respective channels. Therefore, we can omit the connection block in our FPGA model.

Since the *direct-interconnections* are used by neighbor LB's to provide minimum propagation delay and the *long lines* are used by signals which must travel long distances (i.e., global clock), these interconnection resources are not considered in the global routing. Hence, our FPGA model for global routing considers only the LB's, SB's and channel segments. An FPGA can be modeled as a two dimensional array of LB's which are connected to the vertical and horizontal channel segments, and SB's which make connections between

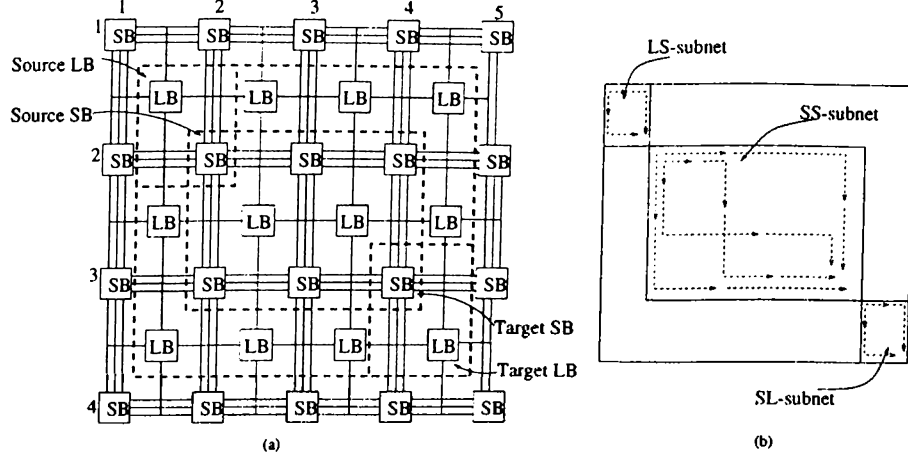


Figure 3.8. (a) The routing area of the two-pin net and its subnets, (b) The possible routes for each subnets

the horizontal and vertical channel segments (Fig. 3.7).

In this work, we divide all multi-pin nets into two-pin nets using minimum spanning tree algorithm [19] as in LocusRoute. Hence, a net refers to a two-pin net here, and hereafter. Consider the possible routings for a two-pin net with a Manhattan distance of $d_h + d_v$ where d_h and d_v denote the horizontal and vertical distances, respectively, between the two pins of the net on the LB grid. The routing area of this net is restricted to a $(d_h + 1) \times (d_v + 1)$ LB grid as shown in Fig. 3.8.a. Then, the shortest distance routing of this net can be decomposed into three *independent* routings as follows. Each pin of this net has only one neighbor SB in the optimal routing area. Hence, each pin can be connected to its unique neighbor SB either through a horizontal or a vertical channel segment (Fig. 3.8). Meanwhile, the optimal routing area for the connection of these two unique SB's is restricted to a $d_h \times d_v$ SB grid embedded in the LB grid (Fig. 3.8). Hence, by exploiting this fact, we further subdivide each net into three two-pin subnets referred here as *LS*, *SS* and *SL* subnets (Fig. 3.8.b). Here, *LS* and *SL* subnets represent the LB-to-SB and SB-to-LB connections, respectively, and *SS* subnets represent the SB-to-SB connection for a particular net. Therefore, we consider only two possible routings for both *LS* and *SL* subnets and $d_h + d_v - 2$ possible one or two bend routings for *SS* subnets for routing the original net.

We define an FPGA graph $\mathbf{F}(L, S, C)$ for modeling the global routing problem in FPGAs. This graph is a $P \times Q$ two-dimensional mesh where L , S and

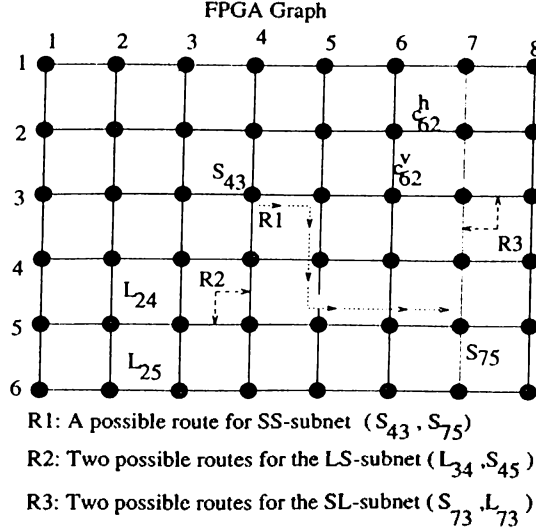


Figure 3.9. The Cost Graph for FPGA model

C denote the set of LB's, SB's and channel segments, respectively. Here, P and Q is the number of horizontal and vertical channels in the FPGA. Each grid point (vertex) s_{pq} of the mesh represents the SB at horizontal channel p and vertical channel q . Each cell L_{pq} of the mesh represents the LB which is adjacent to four SB's s_{pq} , $s_{p,q+1}$, $s_{p+1,q+1}$ and $s_{p+1,q}$. Edges are labeled such that the horizontal (vertical) edge c_{pq}^h (c_{pq}^v) corresponds to the channel segment between the two consecutive SB's s_{pq} and $s_{p,q+1}$ ($s_{p+1,q}$) on the horizontal (vertical) channel p (q), respectively. Figure 3.9 displays a 8×6 sample FPGA graph. Then, the pins of the *LS/SL* and *SS* type subnets are assigned to the respective cell-vertex and vertex-vertex pairs of the graph as is in mentioned earlier.

The global routing problem reduces to searching for most uniform possible distribution of the routes for these subnets. The uniform distribution of the routes is expected to increase the likelihood of finding a feasible routing in the following detailed routing phase. Hence, we need to define an objective function which rewards *balanced* routings. We associate weights with the edges of FPGA graph in order to simplify the computation of the balance quality of a given routing. The weight w_{pq}^h (w_{pq}^v) of a horizontal (vertical) edge c_{pq}^h (c_{pq}^v) denotes the density of the respective channel segment. Here, the density of a channel segment denotes the total number of nets passing through that segment for a given routing. Using this model, we can express the balance

quality B of a given routing \mathbf{R} as

$$B(\mathbf{R}) = \sum_{p=1}^P \sum_{q=1}^{Q-1} (w_{pq}^h(\mathbf{R}))^2 + \sum_{q=1}^Q \sum_{p=1}^{P-1} (w_{pq}^v(\mathbf{R}))^2 \quad (3.2)$$

As is seen in Eq. (3.2), each channel segment contributes the square of its density to the objective function thus penalizing imbalanced routing distributions. Hence, the global routing problem reduces to the minimization of the objective function given in Eq. (3.2).

Chapter 4

MFA SOLUTION FOR GLOBAL ROUTING IN FPGA

This chapter investigates the routing problem in Static RAM Field Programmable Gate Arrays (FPGA's) implementing the non-segmented (Xilinx based) network [27]. The architecture model of FPGA used for formulation and Mean Field Annealing formulation for global routing problem are given in this chapter. Details of experiments, the circuits used in experiments and results are shown at the end of this chapter.

4.1 MFA Formulation of Global Routing

The MFA algorithm is derived by analogy to *Ising* and *Potts* models which are used to estimate the state of a system of particles, called spins, in thermal equilibrium. In Ising model, spins can be in one of the two states represented by 0 and 1, whereas in Potts model they can be in one of the K states. All *LS/SL* subnets are represented by Ising spins since they have only two possible routes. In Ising spin encoding of each *LS/SL* subnet m , $u_m = 1$ (0) denotes that the LB-to-SB or SB-to-LB routing is achieved through a single horizontal (vertical) channel segment. Each *SS* subnet n having $K_n \geq 2$ possible routes is represented by a K_n -state Potts spin. The states of a K_n -state Potts spin is represented using a K_n dimensional vector

$$\mathbf{v}_n = [v_{n1}, \dots, v_{nr}, \dots, v_{n,K_n}]^t \quad (4.1)$$

where “ t ” denotes the vector transpose operation. Each Potts spin \mathbf{v}_n is allowed to be equal to one of the principal unit vectors $\mathbf{e}_1, \dots, \mathbf{e}_r, \dots, \mathbf{e}_{K_n}$, and can not take any other value. Principal unit vector \mathbf{e}_r is defined to be a vector which has all its components equal to 0 except its r 'th component which is equal to 1. Potts spin \mathbf{v}_n is said to be in state r if $\mathbf{v}_n = \mathbf{e}_r$. Hence, a K_n -state Potts spin \mathbf{v}_n is composed of K_n two state variables $v_{n1}, \dots, v_{nr}, \dots, v_{nK_n}$, where $v_{nr} \in \{0, 1\}$, with the following constraint

$$\sum_{r=1}^{K_n} v_{nr} = 1 \quad (4.2)$$

If Potts spin n is in state r (i.e., $v_{nr} = 1$ for $1 \leq r \leq K_n$) we say that the corresponding net n is routed by using the route r .

In the MFA algorithm, the aim is to find the spin values minimizing the energy function of the system. In order to achieve this goal, the average (expected) values $\langle u_m \rangle$ and $\langle \mathbf{v}_n \rangle = [\langle v_{n1} \rangle, \dots, \langle v_{nr} \rangle, \dots, \langle v_{nK_n} \rangle]^t$ of all Ising and Potts spins, respectively, are computed and iteratively updated until the system stabilizes at some fixed point. Note that for each Ising spin m , $u_m \in \{0, 1\}$, i.e., u_m can take only two values 0 and 1, whereas $\langle u_m \rangle \in [0, 1]$, i.e., $\langle u_m \rangle$ can take any real value between 0 and 1. Similarly, for each Potts spin n , $v_{nr} \in \{0, 1\}$ whereas $\langle v_{nr} \rangle \in [0, 1]$. When the system is stabilized, $\langle u_m \rangle$ and $\langle v_{nr} \rangle$ values are expected to converge to either 0 or 1 with the constraints $\sum_{r=1}^{K_n} \langle v_{nr} \rangle = 1$ for the Potts spins.

In order to construct an energy function it is helpful to associate the following meaning to the values $\langle u_m \rangle$ for LS/SL subnets.

$$\begin{aligned} \langle u_m \rangle &= \mathcal{P}(\text{subnet } m \text{ is routed by using the horizontal channel segment}) \\ 1 - \langle u_m \rangle &= \mathcal{P}(\text{subnet } m \text{ is routed by using the vertical channel segment}) \end{aligned}$$

That is, $\langle u_m \rangle$ and $1 - \langle u_m \rangle$ denote the probabilities of finding Ising spin m at states 1 and 0, respectively. In other words, $\langle u_m \rangle$ and $1 - \langle u_m \rangle$ denote the probabilities of routing subnet m through a single horizontal and vertical channel segment, respectively. Similarly, for SS subnets represented with Potts spins

$$\langle v_{nr} \rangle = \mathcal{P}(\text{subnet } n \text{ is routed through route } r) \quad \text{for } 1 \leq r \leq K_n \quad (4.3)$$

That is, $\langle v_{nr} \rangle$ denotes the probability of finding Potts spin at state r for $1 \leq r \leq K_n$. In other words, $\langle v_{nr} \rangle$ denotes the probability of routing net n through

route r . Here and hereafter, u_m and v_{nr} will be used to denote the respective expected values ($\langle u_m \rangle$ and $\langle v_{nr} \rangle$, respectively) for the sake of simplicity. Now, we formulate the total density cost of global routing problem as an energy term

$$\begin{aligned}
 E_B(\mathbf{U}, \mathbf{V}) &= \sum_{p=1}^P \sum_{q=1}^{Q-1} [w_{pq}^h(\mathbf{U}) + w_{pq}^h(\mathbf{V})]^2 + \sum_{q=1}^Q \sum_{p=1}^{P-1} [w_{pq}^v(\mathbf{U}) + w_{pq}^v(\mathbf{V})]^2 \quad (4.4) \\
 \text{where} \quad w_{pq}^h(\mathbf{U}) &= \sum_{m \ni c_{pq}^h} u_m \quad \text{and} \quad w_{pq}^h(\mathbf{V}) = \sum_{n \ni c_{pq}^h} \sum_{r \in R_n, r \ni c_{pq}^h} v_{nr} \\
 w_{pq}^v(\mathbf{U}) &= \sum_{m \ni c_{pq}^v} (1 - u_m) \quad \text{and} \quad w_{pq}^v(\mathbf{V}) = \sum_{n \ni c_{pq}^v} \sum_{r \in R_n, r \ni c_{pq}^v} v_{nr}
 \end{aligned}$$

where $\mathbf{U} = \{u_1, u_2, \dots\}$ and $\mathbf{V} = \{v_1, v_2, \dots\}$ represent the sets of Ising and Potts spins corresponding to the *LS/SL* and *SS* subnets, respectively. For *LS/SL* subnets, “ $m \ni c_{pq}$ ” denotes “for each *LS/SL* subnet m whose pair of pins share the horizontal or vertical channel segment c_{pq} ”. For *SS* subnets “ $n \ni c_{pq}$ ” denotes “for each *SS* subnet n whose routing area contains the horizontal and vertical channel c_{pq} ”. Furthermore, “ $r \in R_n, r \ni c_{pq}$ ” denotes “for each possible route r of *SS* subnet n which passes through the horizontal or vertical channel segment c_{pq} ”. Here, $w_{pq}(\mathbf{U})$ and $w_{pq}(\mathbf{V})$ represent the *probabilistic densities* of the horizontal or vertical channel segment c_{pq} for the current routing states of *LS/SL* and *SS* subnets, respectively. Hence, $w_{pq}(\mathbf{U}, \mathbf{V}) = w_{pq}(\mathbf{U}) + w_{pq}(\mathbf{V})$ represents the total probabilistic density of horizontal or vertical channel segment c_{pq} for the overall current routing state.

Mean field theory equations, needed to minimize the energy function E_B , can be derived as

$$\begin{aligned}
 \phi_m(\mathbf{U}, \mathbf{V}) &= E_B(\mathbf{U}, \mathbf{V})|_{u_m=0} - E_B(\mathbf{U}, \mathbf{V})|_{u_m=1} \\
 &= -2[w_{pq}^h(\mathbf{U}, \mathbf{V}) - w_{pq}^v(\mathbf{U}, \mathbf{V}) - 2(u_m - 0.5)] \quad (4.5) \\
 \text{where} \quad c_{pq}^h, c_{pq}^v &\in m
 \end{aligned}$$

for an Ising spin m and

$$\begin{aligned}
 \phi_{nr}(\mathbf{U}, \mathbf{V}) &= E_B(\mathbf{U}, \mathbf{V})|_{v_n=0} - E_B(\mathbf{U}, \mathbf{V})|_{v_n=r} \quad (4.6) \\
 &= -2\left[\sum_{c_{pq}^h \in r} (w_{pq}^h(\mathbf{U}, \mathbf{V}) - v_{nr}) + \sum_{c_{pq}^v \in r} (w_{pq}^v(\mathbf{U}, \mathbf{V}) - v_{nr}) \right] \\
 &\text{for } 1 \leq r \leq K_n
 \end{aligned}$$

for a Potts spin n , respectively. Mean field values ϕ_m and ϕ_{nr} can be interpreted as the increases in the energy function $E_B(\mathbf{U}, \mathbf{V})$ when Ising and Potts spins m and n are assigned to states 1 and r , respectively. Hence, $-\phi_m$ and $-\phi_{nr}$

may be interpreted as the decreases in the overall solution qualities by routing *LS/SL* and *SS* subnets m and n through the horizontal channel and route r , respectively. Then, u_m and v_{nr} values are updated such that probabilities of routing subnets m and n through horizontal channel and route r increase with increasing mean field values ϕ_m and ϕ_{nr} as follows:

$$u_m = \frac{e^{\phi_m/T}}{1 + e^{\phi_m/T}} \quad (4.7)$$

$$v_{nr} = \frac{e^{\phi_{nr}/T}}{\sum_{k=1}^{K_n} e^{\phi_{nk}/T}} \quad \text{for } r = 1, 2, \dots, K_n \quad (4.8)$$

respectively.

After the mean field equations (Eqs. (4.5-4.6)) are derived, the MFA algorithm can be summarized as follows. First, an initial high temperature spin average is assigned to each spin, and an initial temperature T is chosen. Each u_m value is initialized to $0.5 \pm \delta_m$ and each v_{nr} value is assigned to $1/K_n \pm \delta_{nr}$ where δ_m and δ_{nr} denote randomly selected small disturbance values. Note that $\lim_{T \rightarrow \infty} u_m = 0.5$ and $\lim_{T \rightarrow \infty} v_{nr} = 1/K_n$. In each MFA iteration, the mean field effecting a randomly selected spin is computed using either Eq. (4.5) or Eq. (4.6). Then, the average of this spin is updated using either Eq. (4.7) or Eq. (4.8). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. The system is observed after each spin update in order to detect the convergence to an equilibrium state for a given temperature. If energy function E_B does not decrease in most of the successive spin updates, this means that the system is stabilized for that temperature. Then, T is decreased according to a cooling schedule, and iterative process is re-initialized. At the end of this cooling schedule, each Ising spin m is set to state 1 if $u_m \geq 0.5$ or to state 0, otherwise. Similarly, maximum element in each Potts spin vector is set to 1 and all other element are set to 0. Then, the resulting global routing is decoded as mentioned earlier.

4.2 Implementation

The performance of the proposed MFA algorithm for the global routing problem is evaluated in comparison with the well-known LocusRoute algorithm [24].

The MFA global router is implemented efficiently as described in Section 4.1. Average of each Ising spin m is initialized by randomly selecting u_m^{init}

in the range $0.45 \leq u_m \leq 0.55$. Similarly, average of each Potts spin n is initialized by randomly selecting K_n v_{nr} values in the range $0.9/K_n \leq v_{nr} \leq 1.1/K_n$ and normalizing $v_{nr}^{init} = v_{nr} / \sum_{k=1}^{K_n} v_{nk}$ for $r = 1, 2, \dots, K_n$. Note that random selections are achieved by using uniform distribution in the given ranges.

The initial temperature parameter used in mean field computation is estimated using the initial spin averages values. Selection of initial temperature parameters T_0 is crucial to obtain good routing. In previous applications of MFA, it is experimentally observed that spin averages tend to converge at a critical temperature. Although there are some methods proposed for the estimation of critical temperature, we prefer an experimental way for computing T_0 which is easy to implement and successful as the results of experiments indicate. We compute the initial average mean field as

$$\phi_{avg}^{init} = \left(\sum_{m=1}^{N_m} \phi_m^{init} + \sum_{n=1}^{N_n} \sum_{k=1}^{K_n} \phi_{nr}^{init} \right) / (N_m + \sum_{n=1}^{N_n} K_n)$$

Note that initial mean field values ϕ_m^{init} and ϕ_{nr}^{init} are computed according to Eqs. (4.5) and (4.6) using initial spin values u_m^{init} and v_{nr}^{init} . Here, N_m and N_n denote the total number of Ising and Potts spins, respectively, where $N = N_m + N_n$ denotes the total number of spins (subnets). Then, initial temperature is computed as $T_0 = C \phi_{avg}^{init}$ where constant C is chosen as 540 for all experiments.

The cooling schedule is an important factor in the performance of MFA global router. For a particular temperature, MFA proceeds for randomly selected unconverged net spin updates until $\Delta E < \epsilon$ for M consecutive iterations respectively where $M = N$ initially and $\epsilon = 0.05$. Average spin values are tested for convergence after each update. For an Ising spin m , if either $u_m \leq 0.05$ or $u_m \geq 0.95$ is detected, then spin m is assumed to converge to state 0 or state 1, respectively. For a Potts spin n , if $v_{nr} \geq 0.95$ is detected for a particular $r = 1, 2, \dots, K_n$, then spin n is assumed to converge to state r . The cooling process is realized in two phases, slow cooling followed by fast cooling, similar to the cooling schedules used for Simulated annealing. In the slow cooling phase, temperature is decreased by $T = \alpha \times T$ where $\alpha = 0.9$ until $T < T_0/1.5$. Then, in the fast cooling phase, M is set to $M/2$, α is set to 0.8. Cooling schedule continues until 90% of the spins converge. At the end of this cooling process, each unconverged Ising spin m is assumed to converge to state 0 or state 1 if $u_m < 0.5$ or $u_m \geq 0.5$, respectively. Similarly, each unconverged Potts spin n is assumed to converge to state r where $v_{nr} = \max\{v_{nk} : k = 1, 2, \dots, K_n\}$. Then, the result is decoded as described in Section 4.1, and the resulting global routing is found.

Table 4.1. MCNC benchmark circuits used in experiments

Benchmarks			
Circuits			
name	number of nets	number of 2-pin nets	FPGA size
<i>9symml</i>	71	259	10x9
<i>too-large</i>	177	519	14x13
<i>apex7</i>	124	300	11x9
<i>example2</i>	197	444	13x11
<i>vda</i>	216	722	16x15
<i>alu2</i>	137	511	14x12
<i>alu4</i>	236	851	18x16
<i>term1</i>	87	202	9x8
<i>C1355</i>	142	360	12x11
<i>C499</i>	142	360	12x11
<i>C880</i>	173	427	13x11
<i>K2</i>	388	1256	21x19
<i>Z03D4</i>	575	2135	26x25
<i>buscntl</i>	145	392	12x11
<i>dram fsm</i>	389	1422	22x21
<i>dma</i>	197	771	17x15
<i>z03</i>	575	2135	26x25

The LocusRoute algorithm is implemented as in [24]. As the LocusRoute depends on rip-up and reroute method, LocusRoute is allowed to reroute the circuits 5 times. No bend reduction has been done as in [3]. Both algorithms are implemented in the C programming language.

4.3 Experimental Results

This section presents experimental performance evaluation of the proposed MFA algorithm in comparison with LocusRoute and Simulated Annealing (SA) algorithm. All algorithms are tested for the global routing of thirteen *ACM SIGDA Design Automation* benchmarks (MCNC) and four famous FPGA benchmark circuits on SUN SPARC 10 . The Table 4.1 illustrates the properties of these benchmark circuits.

These three algorithms yield the same total wiring length for global routing since two or less bend routing scheme is adopted in all of them. Necessary design automation process such as technology mapping and placement are done in University of Toronto by using Chortle technology mapper [11] and XAltor placement tools.

Table 4.2. The Global Router results

Circuit	MFA			PGA			SA		
	Cost	Dens	time	Cost	Dens	time	Cost	Dens	time
<i>9symm1</i>	1.0	12.0	0.36	1.032	14	0.00	1.000	12.0	20.64
<i>toolarge</i>	1.0	16.0	0.88	1.071	17	0.06	1.003	16.0	113.90
<i>apez7</i>	1.0	14.0	0.42	1.073	16	0.00	0.935	14.0	31.46
<i>example2</i>	1.0	15.0	0.64	1.097	16	0.02	0.856	15.0	76.54
<i>vda</i>	1.0	17.0	0.42	1.055	18	0.10	1.002	17.0	207.80
<i>alu2</i>	1.0	17.0	0.30	1.080	17	0.02	0.928	17.0	91.44
<i>alu4</i>	1.0	17.0	0.68	1.073	19	0.10	0.966	17.0	288.78
<i>term1</i>	1.0	14.0	0.34	1.093	14	0.00	0.921	14.0	13.28
<i>C1355</i>	1.0	13.0	0.56	1.119	15	0.00	1.000	13.6	50.36
<i>C499</i>	1.0	15.0	0.48	1.075	16	0.00	1.003	15.0	44.58
<i>C880</i>	1.0	15.4	0.68	1.065	17	0.04	0.933	16.8	74.40
<i>k2</i>	1.0	20.2	0.94	1.038	22	0.20	0.952	20.0	712.10
<i>z03D4</i>	1.0	17.0	2.34	1.117	18	0.30	1.000	17.0	1821.12
<i>busectl</i>	1.0	13.0	0.42	1.050	13	0.00	0.998	13.0	54.92
<i>dram fsm</i>	1.0	15.0	1.94	1.073	18	0.20	0.999	15.0	763.02
<i>dma</i>	1.0	15.0	1.96	1.084	16	0.10	0.972	15.0	216.80
<i>z03</i>	1.0	20.0	2.10	1.119	21	0.30	1.000	20.0	1837.86

Table 4.2 illustrates the performance results of these three algorithms for the benchmark circuits. The MFA algorithm is executed 10 times for each circuit starting from different, randomly chosen initial configurations. The results given for the MFA algorithm in Table 4.2 illustrate the average of these executions. Global routing cost values of the solutions found by both algorithms are computed using Eq. (3.2) and then normalized with respect to those of MFA. In Table 4.2, maximum channel density denotes the number of routes assigned to the maximally loaded channels. That is, it denotes the minimum number of tracks required in a channel for 100% routability.

As is seen in Table 4.2, global routing costs of the solutions found by MFA are 3.1%-10.5% better than those of LocusRoute. As is also seen in this table, maximum channel density requirements of the solutions found by MFA are less than those of LocusRoute in almost all circuits except *alu2* and *term1*. Both algorithms obtain the same maximum channel density for these two circuit.

How the global router distributes the channel densities, how the global router decrease the maximum channel densities and how detailed router completes the routing are some important metrics to measure the quality of the global routers. The propagation net delays, number of switch used , number of tracks in a channel are considered in comparison of global routers after completion of routing. The channel densities distribution affects on the number of tracks and switch also the propagation delay (because of number of switches) of the nets. In next paragraphs, the results of global routes are given in terms

Table 4.3. The SEGA detailed routing results in area optimization mode

Circuit	Routing Info.						Delay Info.				
	Total Segment			Shared			Avg. Delay			Max. Delay	
	MFA	PGA	Imp	MFA	PGA		MFA	PGA	Imp	MFA	PGA
9symml	674	711	5.20	42	85		5.06	5.56	9.01	63.38	57.97
toolg	1803	1951	7.59	47	114		13.83	15.10	8.45	125.48	122.80
apex7	960	1026	6.43	36	63		9.88	10.64	7.15	70.97	77.65
exp2	1775	1893	6.23	42	56		10.08	11.98	15.86	101.31	121.88
vda	2760	2950	6.44	70	176		18.67	20.58	9.30	140.77	170.36
alu2	1580	1674	5.62	36	129		9.82	9.61	-2.12	129.24	110.30
alu4	3183	3424	7.04	67	203		16.58	17.08	2.93	153.88	163.30
term1	602	638	5.64	21	47		9.57	9.60	0.32	74.81	70.50
C1355	1299	1347	3.56	27	82		12.17	13.15	7.50	121.01	118.12
C499	1242	1296	4.17	37	82		11.64	12.02	3.15	79.75	94.46
C880	1575	1670	5.69	38	91		14.83	15.36	3.48	111.58	115.72
K2	5980	6323	5.42	88	306		25.77	27.54	6.43	244.35	229.54
Z03D4	7125	7700	7.47	227	555		12.75	13.60	6.26	190.62	191.65
bus-ctrl	1128	1213	7.01	43	94		7.94	8.57	7.28	104.36	126.24
dram-fsm	4267	4648	8.20	174	403		6.19	6.68	7.35	140.61	157.05
dma	2300	2545	9.63	94	214		15.17	16.58	8.53	200.82	194.71
z03	7161	7870	9.01	267	533		13.05	14.40	9.39	193.18	192.93

of these metrics. The balance cost of SA and MFA global routers are not very different but the execution time of SA is 250 times longer than the MFA on the averages for all circuit.

The detailed router used in this experiments is called SEGA [20], for Segment Allocator, and was developed specifically for SRAM based FPGA's. The input of SEGA is a netlist of two point connections, which is output of the global router. To route the connections, SEGA allocates wire segments according to cost function, basing its decisions on either of two goals: optimize for area or optimize for speed. For area optimization, only routability of the circuit is considered, which means the cost function focuses only on the task of successfully routing 100% of the connections in a circuit. In delay optimization, SEGA selects the routes that have the best speed performance. The following assumption are done in experiments. All routing channels have an equal number of tracks. The flexibility of the channel blocks are equal to number of tracks. (Each logic pin can connect to a channel with all tracks) The LocusRoute global routing algorithm used in PgaRoute global router (PGA). For further part of this chapter, PGA global router are used for LocusRoute algorithm [23].

The SEGA detailed router routes the nets by considering either area optimization or speed optimization criteria. Therefore all circuits are tested according to these two optimization criteria, separately. The output of MFA and

Table 4.4. The SEGA detailed routing results in speed optimization mode

Circuit	Routing Info.						Delay Info.			
	Total Segment			Shared		Avg. Delay			Max. Delay	
	MFA	PGA	Imp	MFA	PGA	MFA	PGA	Imp	MFA	PGA
9symml	653	649	-0.62	63	147	5.07	5.28	3.94	56.46	48.67
toolg	1776	1822	2.52	74	243	13.34	13.06	-2.17	128.56	106.00
apex7	942	952	1.05	54	137	9.73	9.86	1.28	70.97	63.32
exp2	1746	1762	0.91	71	187	10.01	10.81	7.40	95.27	98.10
vda	2704	2774	2.52	126	352	19.07	19.10	0.17	148.30	164.71
alu2	1533	1542	0.58	83	261	9.46	9.56	1.07	127.29	128.45
alu4	3132	3193	1.91	118	434	16.17	16.29	0.76	145.32	147.41
term1	591	592	0.17	32	93	9.74	8.13	-19.82	76.82	46.33
C1355	1277	1269	-0.63	49	160	12.34	11.69	-5.59	126.73	98.27
C499	1225	1222	-0.25	54	156	11.66	10.72	-8.81	81.49	83.71
C880	1552	1567	0.96	61	194	14.39	14.01	-2.73	106.94	106.06
K2	5900	5995	1.58	168	634	27.05	26.50	-2.10	262.23	210.25
Z03D4	6965	7664	9.12	437	1191	12.42	12.34	-0.65	167.32	169.05
bus-cntl	1112	1114	0.18	59	193	8.03	7.95	-1.04	95.93	86.24
dram-fsm	4155	4305	3.48	286	746	6.05	6.61	8.54	140.61	146.57
dma	2243	2350	4.55	151	409	14.89	15.40	3.30	203.74	181.06
z03	6953	7205	3.50	475	1198	12.65	13.27	4.69	172.34	173.38

PGA global routers was used as a input of the detailed router. After that SEGA detailed router was executed in two different mode (area and speed optimization mode) for each benchmark circuit. The results of SEGA detailed router gives information about *routing* which contains total number of segment, shared segment and minimum channel width for 100% routing, and *propagation delay* which contains average and maximum delay of the nets. Therefore, quality of MFA and PGA global routers are compared by considering these routing and delay information.

Table 4.3, Table 4.4 and Table 4.5 shows the results of SEGA detailed router whose inputs were constructed by MFA and PGA routers. Table 4.3 represents the results for area optimization mode and Table 4.4 represents the results for speed optimization mode. As seen in Table 4.3, MFA needs less number of segment that PGA for complete routing. There are 3%-9% improvement in total number of segment used in complete routing. Also MFA causes less propagation delay than MFA for all benchmark circuits as in Table 4.3. The average delay for routing are decreased by 3%-15% for MFA according to PGA. If we consider the number of tracks in a channel, MFA needs small channel width in 6 benchmarks, but PGA routes 8 benchmarks with less number of tracks than MFA. For other benchmarks circuit both PGA and MFA need same channel width as seen in Table 4.5 Finally we can say that MFA global router produces better results that PGA global router according to area optimization. Because, MFA can distribute the channel density more that PGA. Also SEGA

Table 4.5. Minimum Channel Width for 100% routing

Circuit	Channel Width (W)			
	Area Opt. Mode		Speed Opt. Mode	
	MFA	PGA	MFA	PGA
9symml	10	9	10	11
toolg	13	11	13	12
apex7	11	13	12	15
exp2	13	17	14	19
vda	13	16	16	16
alu2	13	10	13	12
alu4	14	13	15	15
term1	10	9	11	10
C1355	10	12	12	12
C499	13	11	14	11
C880	12	13	13	14
K2	15	16	19	19
Z03D4	14	14	15	15
bus-cntl	10	10	11	11
dram-fsm	13	11	13	13
dma	11	11	12	13
z03	16	14	16	16

detailed router results in speed optimization mode as in Table 4.4 shows that there are also improvement in both total number of segment, channel width and average delay. But the percent of improvement is less than those of area optimization mode. Note that PGA can cause less maximum delay than MFA for most of circuits.

Also the channel width is important criteria for routing because its affect on the size of FPGAs. In Table 4.5 the minimum number of track (channel width) in a channel are shown for both area and speed optimization mode. As in this table, for some circuits, MFA gives better results but some circuits PGA gives better results, therefore the MFA's and PGA's performance on channel width are very similar.

Figures 4.1 and 4.2 contain visual illustrations as pictures (left) and histograms (right) for the channel density distributions of the solutions found by MFA and LocusRoute, respectively, for the circuit *C1355*. The pictures are painted such that the darkness of each channel increases with increasing channel density. Global routing solutions found by these two algorithms are tested by using SEGA detailed router for FPGA. Figure 4.3 illustrates the results of the SEGA detailed router for the circuit *C1355*

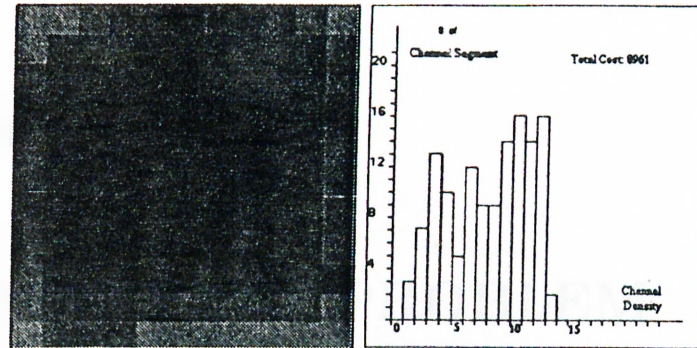


Figure 4.1. Channel density distribution obtained by MFA for the circuit *C1355*

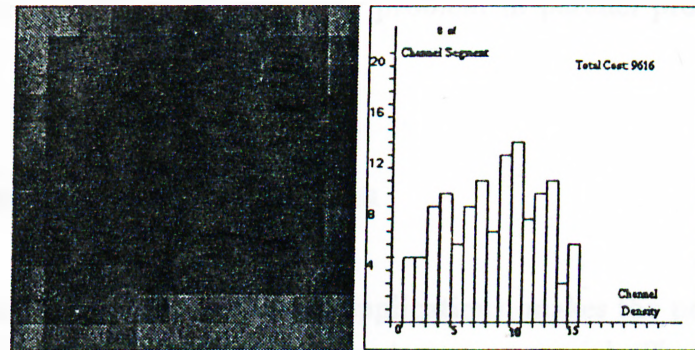


Figure 4.2. Channel density distribution obtained by LocusRoute for the circuit *C1355*

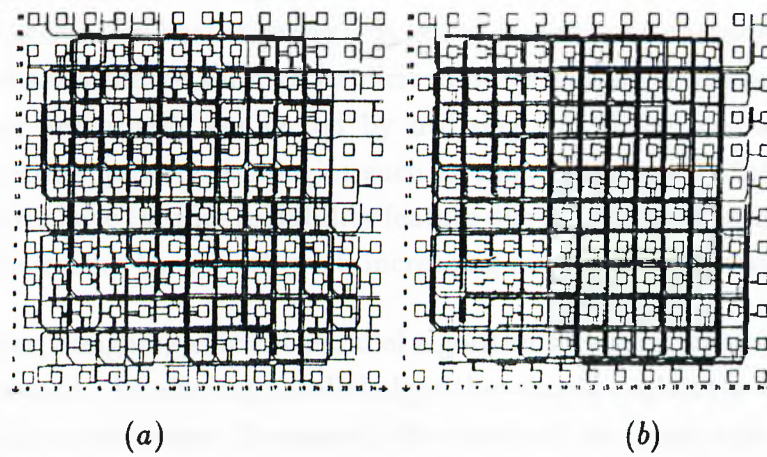


Figure 4.3. SEGA detailed router results of the circuit *C1355* for the global routing solutions obtained by (a) MFA (b) LocusRoute

Chapter 5

THE MAPPING PROBLEM

This chapter introduces the mapping problem in parallel processing and its application.

5.1 The Mapping Problem

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 [4], 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 [13]. Hence, heuristics giving sub-optimal solutions are used to solve the problem [4, 13, 21]. Two distinct approaches have been considered in the context of mapping heuristics, one-phase approaches and two phase approaches. 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 [21]. 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 [21].

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 [10, 17] is an efficient heuristic, originally proposed for the graph bipartitioning problem, which can also be used for clustering [21]. 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 [15].

Simulated Annealing (SA) can also be used as a one phase heuristic for

solving many-to-one mapping problem [15, 28]. Successful applications of SA to the mapping problem is achieved in various works [15, 28]. It has been observed that the quality of the solutions obtained using SA are superior compared with the results of the other heuristics.

5.2 The Model of Mapping Problem

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 represents the interaction pattern among the tasks. Relative computational costs of atomic tasks can be known or estimated prior to the execution of the parallel program. Hence, weights can be associated with the vertices in order to denote the computational costs of the corresponding tasks.

There are some model to model the static task interaction pattern. One of the model is Task Interaction Graph (TIG) model. In the TIG model, interaction patterns are represented by undirected edges between 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 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 [26]. However, there are various classes of problems which can be successfully modeled with the TIG model. For example, iterative solution of systems of equations arising in finite element applications [7, 26] and power system simulations, and VLSI simulation programs [28] are represented by TIGs. In this work, problems which can be represented by the TIG model are addressed.

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 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 by means of *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 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, in software routing, the unit communication cost is linearly proportional to the shortest path distance between the pair of communicating processors. Hence, the communication topology of the multicomputer can be modeled by an undirected complete graph, referred here as Processor Communication Graph (PCG). The nodes of 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 POG and the *routing* scheme utilized for inter-processor communication.

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

A mapping problem instance can be formally represented with 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 G_T 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 G_P , labeled as $(1, 2, \dots, p, q, \dots, K)$, represent the processors of the target multicomputer. Edge weight d_{pq} , for $1 \leq p, q \leq K$ and $p \neq q$, denotes the unit communication cost between processors p and q .

Given an instance of the mapping problem with the TIG $G_T(V, E)$ and the PCG $G_P(P, D)$, the 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 the graph G_P , and minimizes the total interprocessor communication cost (CC)

$$CC = \sum_{(i,j) \in E, M(i) \neq M(j)} e_{ij} d_{M(i)M(j)} \quad (5.1)$$

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

$$CL_p = \sum_{i \in V, M(i)=p} w_i, \quad 1 \leq p \leq K \quad (5.2)$$

of each processor balanced. Here, $M(i) = p$ denotes the label (p) of the processor that task i is mapped to. In Eq. (5.1), each edge (i, j) of the G_T contributes to communication cost (CC), only if vertices i and j are mapped to two different nodes of the G_P , 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 each p , $1 \leq p \leq K$. Computational load balance of the processors can be explicitly included in the cost function using a term which is minimized when all processor loads are equal. Another scheme is to include load balance criteria implicitly in the algorithm.

In Figure 5.1, an example for mapping problem are shown. The TIG graph is in Fig. 5.1.a and a corresponding mapping instance is in Fig. 5.1.b

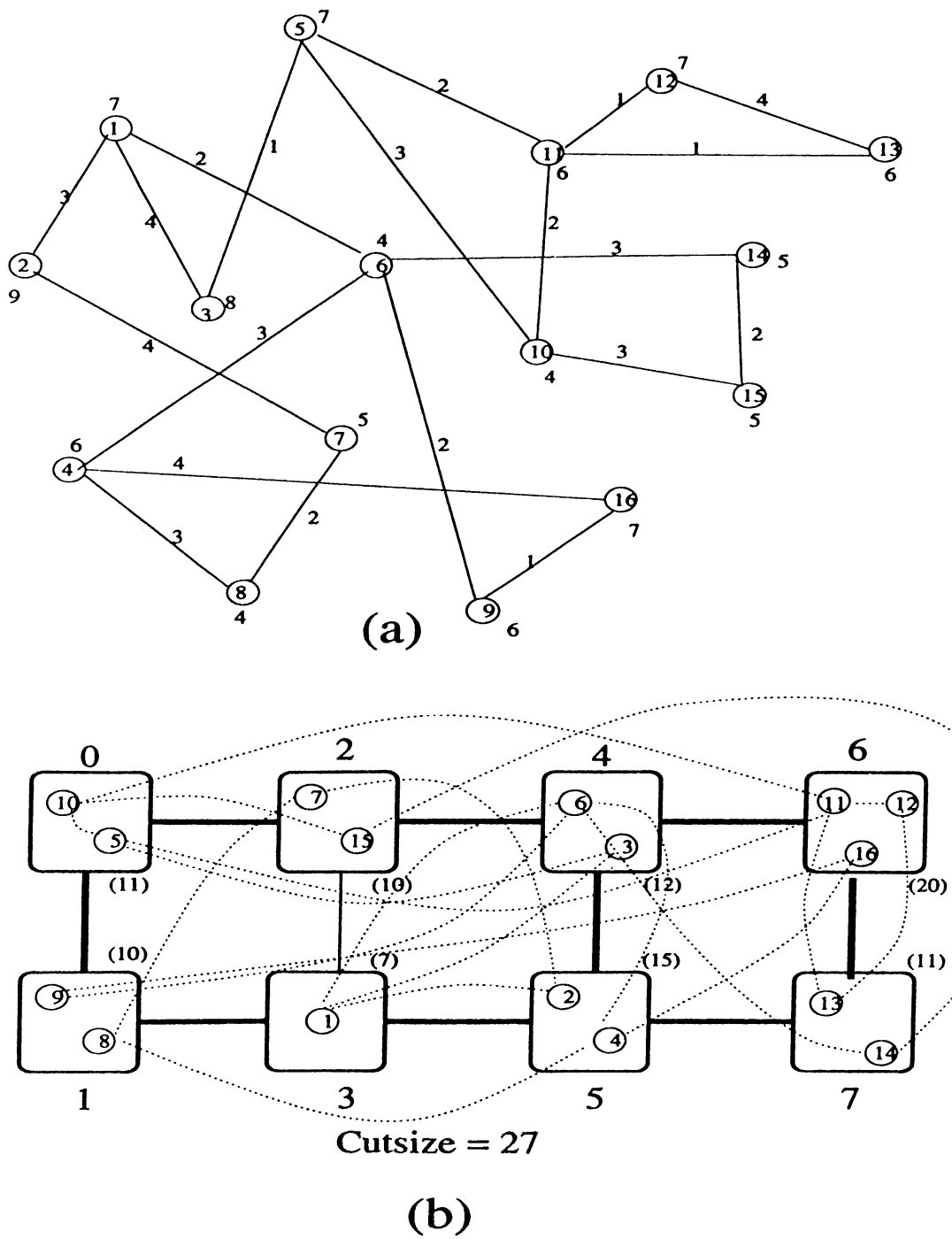


Figure 5.1. An example of mapping problem

Chapter 6

MFA SOLUTION FOR MAPPING

In this chapter, the general MFA formulation and a new efficient MFA formulation for mapping problem in mesh and hypercube type multicomputer are proposed. The experimental results for randomly generated mapping instances and real problem instances are shown at the end of this chapter.

6.1 General MFA Formulation for Mapping Problem

The MFA algorithm is derived by analogy to *Ising* and *Potts* models which are used to estimate the state of a system of particles, called spins, in thermal equilibrium. In Ising model, spins can be in one of the two states represented by 0 and 1, whereas in Potts model they can be in one of the K states. In this work we use the Potts model. In the K state Potts model of S spins, the states of spins are represented using S K -dimensional vectors

$$\mathbf{S}_i = [s_{i1}, \dots, s_{ik}, \dots, s_{iK}]^t \quad \text{for } i = 1, 2, \dots, S.$$

where “ t ” denotes the transpose operation. The spin vector \mathbf{S}_i is allowed to be equal to one of the principal unit vectors $\mathbf{e}_1, \dots, \mathbf{e}_k, \dots, \mathbf{e}_K$, and can not take any other value. Principal unit vector \mathbf{e}_k is defined to be a vector which has all its components equal to 0 except its k 'th component which is equal to 1. Spin \mathbf{S}_i is said to be in state k if $\mathbf{S}_i = \mathbf{e}_k$. Hence, a K -state Potts spin \mathbf{S}_i is composed of K two state variables $\{s_{ik}\}_{k=1}^K$, where $s_{ik} \in \{0, 1\}$, with the following constraint

$$\sum_{k=1}^K s_{ik} = 1 \quad \text{for } i = 1, 2, \dots, S. \quad (6.1)$$

In the general encoding of the mapping problem, each spin vector corresponds to a vertex of the TIG $G(T, I)$. Hence, number of spins vectors is $S = |T| = N$. Dimension K of the spin vectors is equal to the number of processors. If a spin is in state k (i.e., $s_{ik} = 1$) we say that the corresponding task is assigned to processor k .

In the MFA algorithm, the aim is to find the spin values minimizing the energy function of the system. In order to achieve this goal, the average (expected) values $\mathbf{V}_i = \langle \mathbf{S}_i \rangle$ of each spin vector \mathbf{S}_i is computed and iteratively updated until the system stabilizes at some fixed point. Hence, we define

$$\mathbf{V}_i = [v_{i1}, \dots, v_{ik}, \dots, v_{iK}]^t = \langle \mathbf{S}_i \rangle = [\langle s_{i1} \rangle, \dots, \langle s_{ik} \rangle, \dots, \langle s_{iK} \rangle]^t \quad (6.2)$$

That is, $v_{ik} = \langle s_{ik} \rangle$, for $i = 1, 2, \dots, S$ and $k = 1, 2, \dots, K$. Note that, $s_{ik} \in \{0, 1\}$, i.e., s_{ik} can take only two values 0 and 1, whereas $v_{ik} \in [0, 1]$, i.e., v_{ik} can take any real value between 0 and 1. As the system is a Potts glass we have the following constraint similar to Eq. (6.1)

$$\sum_{k=1}^K v_{ik} = 1, \quad \text{for } i = 1, 2, \dots, N \quad (6.3)$$

This constraint guarantees that each Potts spin \mathbf{S}_i is in one of the K states at a time, and each task is mapped to only one processor. In order to construct an energy function it is helpful to associate the following meaning to the values v_{ik} ; $v_{ik} = \mathcal{P}(\text{task } i \text{ is mapped to the processor } k)$ for $i = 1, 2, \dots, N$, and $k = 1, 2, \dots, K$. That is, v_{ik} is the probability of finding spin i at state k . If $v_{ik} = 1$ then spin i is in state k and the corresponding configuration is $\mathbf{S}_i = \mathbf{V}_i$.

Now, we formulate the communication cost of the mapping problem as an energy term

$$\begin{aligned} E^C(\mathbf{V}) &= \sum_{(i,j) \in I} e_{ij} \sum_{k=1}^K \sum_{l \neq k}^K d_{kl} \mathcal{P}(\text{task } i \text{ is mapped to processor } k) \\ &\quad \mathcal{P}(\text{task } j \text{ is mapped to processor } l) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \in \text{Adj}(i)} \sum_{k=1}^K \sum_{l \neq k}^K e_{ij} v_{ik} v_{jl} d_{kl} \end{aligned} \quad (6.4)$$

where $\mathbf{V} = [\mathbf{V}_1, \dots, \mathbf{V}_k, \dots, \mathbf{V}_K]^t$ is the spin average matrix consisting of N K -dimensional spin vectors as its rows. Here, $\text{Adj}(i)$ denotes the set of tasks connected to task i in the given TIG. Minimization of E^C corresponds to the minimization of the communication cost of the mapping problem. Another

term of the energy function is the term for penalizing imbalanced mappings.

$$\begin{aligned}
E^B(\mathbf{V}) &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \mathcal{P}(\text{tasks } i \text{ and } j \text{ are mapped to the same processor}) \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{k=1}^K \mathcal{P}(\text{task } i \text{ is mapped to processor } k) \\
&\quad \mathcal{P}(\text{task } j \text{ is mapped to processor } k) \\
&= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N \sum_{k=1}^K v_{ik} v_{jk} w_i w_j
\end{aligned} \tag{6.5}$$

This triple summation term computes the summation of the inner products of the weights of the tasks assigned to individual processors. Global minimum of this term occurs when equal amounts of task weights are assigned to each processor. If there is an imbalance in the mapping, E^B term increases with the square of the amount of the imbalance, penalizing imbalanced mappings. The total energy function E is defined in terms of E^C and E^B as

$$E(\mathbf{V}) = E^C(\mathbf{V}) + \beta E^B(\mathbf{V}) \tag{6.6}$$

where parameter β is introduced to maintain a balance between the two optimization objectives of the mapping problem. Mean field theory equations, needed to minimize the energy function E , can be derived as

$$\phi_{ik} = -\frac{\partial E(\mathbf{V})}{\partial v_{ik}} = -\sum_{j \in \text{Adj}(i)} \sum_{l \neq k}^K e_{ij} d_{kl} v_{jl} - \beta \sum_{j \neq i}^N w_i w_j v_{jk} \tag{6.7}$$

The quantity ϕ_{ik} represents the k 'th element of the *mean field* vector effecting on spin k . Using the mean field values ϕ_{ik} , average spin values v_{ik} can be updated using the Boltzmann distribution as

$$v_{ik} = \frac{e^{\phi_{ik}/T}}{\sum_{l=1}^K e^{\phi_{il}/T}} \quad \text{for } i = 1, 2, \dots, N, k = 1, 2, \dots, K \tag{6.8}$$

where T is the temperature parameter which is used to relax the system iteratively. Equation (6.8) handles the constraints given in Eq. (6.3) thus enforcing each Potts spin S_i to be in one of the K state when they converge.

In Eq. (6.7), the first and second summation terms represent the increases in the total communication and imbalance costs, respectively, by mapping task i to processor k . Hence, $-\phi_{ik}$ may be interpreted as the decrease in the overall solution quality by assigning task i to processor k . Then, in Eq. (6.8), v_{ik} is updated such that the probability of mapping task i to processor k increases with

increasing mean field ϕ_{ik} . After the mean field theory equations are derived (Eq. (6.7), Eq. (6.8)), MFA algorithm can be summarized as follows. First an initial, high temperature, spin average is assigned to each spin, and an initial temperature is chosen. At each temperature, starting with initial spin averages, the mean field vector effecting on a randomly selected spin is computed using Eq. (6.7). Then, spin average vector is updated using Eq. (6.8). This process is repeated for a random sequence of spins until the system is stabilized for the current temperature. Then, T is decreased according to the cooling schedule, and iterative process is re-initiated. In [6] we have proposed an efficient implementation scheme which asymptotically reduces the complexity of a MFA iteration to $\Theta(d_{avg}K + K^2)$ where d_{avg} denotes the average vertex degree in the TIG.

6.2 Interconnection-Topology Specific MFA Formulation for Mapping

In this section, we proposed efficient Mean Field Annealing formulation for Mesh-connected and Hypercube-connected architecture.

6.2.1 MFA formulation for Mesh-Connected Architectures

Consider a P by Q two-dimensional mesh-connected architecture with P rows and Q columns. The encoding in the general MFA formulation summarized in Section 6.1 necessitates $N \times K = N \times P \times Q$ variables for the problem representation. In this section, we propose a MFA formulation for the mesh-connected architectures which exploits the conventional routing scheme in mesh interconnection topologies to introduce a much more efficient encoding scheme. Note that, the communication distance between any two processors is equal to the Manhattan distance between those two processors on the processor grid. Hence, the unit communication cost between any two processors can be expressed as the sum of two components: horizontal and vertical communication costs. Horizontal and vertical unit communication costs are equal to the column and row distances between the processor pairs, respectively. Thus, any edge $(i, j) \in I$

with weight e_{ij} of the TIG will contribute

$$E_{ij}^C = E_{ij}^h + E_{ij}^v = e_{ij} \times |\text{column}(i) - \text{column}(j)| + e_{ij} \times |\text{row}(i) - \text{row}(j)| \quad (6.9)$$

to the total communication cost, where $\text{row}(i)$ and $\text{column}(i)$ denote the row and column indices of the processor that task i is mapped to and $|\cdot|$ denotes the absolute value function. Here, E_{ij}^v and E_{ij}^h denote the horizontal and vertical communication costs due to edge $(i, j) \in I$ of the TIG. Hence, the row and column mappings of each task are sufficient for efficient computation of the interprocessor communication cost in mesh-connected architectures.

Encoding

In the proposed encoding, we use two Potts spins of dimensions P and Q for each vertex (task) of the TIG. Spins of dimensions P and Q are used to encode the row and column mappings of the tasks, respectively. Note that this encoding also constructs a one-to-one mapping between the configuration space of the problem domain and the spin domain. However, it is much more efficient since it uses a total of $N \times (P + Q)$ two-state variables instead of $N \times P \times Q$ two state variables of the general encoding. Spins with dimensions P and Q are called row and column spins which are labeled as $\mathbf{S}_i^r = [s_{i1}^r, \dots, s_{ip}^r, \dots, s_{iP}^r]^t$ and $\mathbf{S}_i^c = [s_{i1}^c, \dots, s_{iq}^c, \dots, s_{iQ}^c]^t$, respectively, for $i = 1, 2, \dots, N$. If a row (column) spin is in state p (q) we say that the corresponding task is mapped to row p (column q). Hence, $s_{ip}^r = 1$ ($s_{iq}^c = 1$) means that task i is mapped to row p (column q) of the mesh. That is, if $s_{ip}^r = 1$ and $s_{iq}^c = 1$, this means that task i is mapped to processor pq in the mesh. Here, processor pq identifies the processor at row p and column q of the mesh.

Energy Function Formulation

The following spin average vectors are defined for the sake of energy function formulation.

$$\begin{aligned} \mathbf{V}_i^r &= [v_{i1}^r, \dots, v_{ip}^r, \dots, v_{iP}^r]^t = \langle \mathbf{S}_i^r \rangle = [\langle s_{i1}^r \rangle, \dots, \langle s_{ip}^r \rangle, \dots, \langle s_{iP}^r \rangle]^t \\ \mathbf{V}_i^c &= [v_{i1}^c, \dots, v_{iq}^c, \dots, v_{iQ}^c]^t = \langle \mathbf{S}_i^c \rangle = [\langle s_{i1}^c \rangle, \dots, \langle s_{iq}^c \rangle, \dots, \langle s_{iQ}^c \rangle]^t \end{aligned}$$

Note that, $s_{ip}^r, s_{iq}^c \in \{0, 1\}$, i.e., s_{ip}^r and s_{iq}^c are discrete variables taking only two values 0 and 1, whereas $v_{ip}^r, v_{iq}^c \in [0, 1]$, i.e., v_{ip}^r and v_{iq}^c are continuous variables

taking any real value between 0 and 1. As the system is a Potts glass we have the following constraints similar to Eq. (6.3)

$$\sum_{p=1}^P v_{ip}^r = 1, \quad \sum_{q=1}^Q v_{iq}^c = 1, \quad (6.10)$$

These constraints guarantee that each Potts spin S_i^r (S_i^c) is in one of the P (Q) states at a time, and each task is assigned to only one row (column) for the proposed encoding. In order to construct an energy function it is helpful to associate the following meanings to the v_{ip}^r and v_{iq}^c values,

$$\begin{aligned} v_{ip}^r &= \mathcal{P}(\text{task } i \text{ is mapped to one of the processor in row } p), \\ v_{iq}^c &= \mathcal{P}(\text{task } i \text{ is mapped to one of the processor in column } q) \end{aligned} \quad (6.11)$$

for $i = 1, 2, \dots, N$, $p = 1, 2, \dots, P$ and $q = 1, 2, \dots, Q$. That is, v_{ip}^r (v_{iq}^c) denotes the probability of finding row (column) spin i in row p (column q). Formulation of horizontal communication cost due to edge (i, j) of the TIG as an energy term is:

$$\begin{aligned} E_{(i,j)}^h &= e_{ij} \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q (l-k) \\ &\quad \times \{ \mathcal{P}(\text{tasks } i \text{ and } j \text{ are mapped to columns } k \text{ and } l, \text{ respectively}) + \\ &\quad \mathcal{P}(\text{tasks } j \text{ and } i \text{ are mapped to columns } k \text{ and } l, \text{ respectively}) \} \\ &= e_{ij} \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q (l-k) (v_{ik}^c v_{jl}^c + v_{jk}^c v_{il}^c) \end{aligned} \quad (6.12)$$

Similarly, energy formulation for the vertical communication cost due to edge (i, j) is

$$E_{(i,j)}^v = e_{ij} \sum_{k=1}^{P-1} \sum_{l=k+1}^P (l-k) (v_{ik}^r v_{jl}^r + v_{jk}^r v_{il}^r) \quad (6.13)$$

The derivation of the mean field theory equation using the formulation of the energy terms $E_{(i,j)}^h$ and $E_{(i,j)}^v$ given in Eqs. (6.12) and (6.13) results in substantially complex expressions. Hence, we simplify the expressions for $E_{(i,j)}^h$ and $E_{(i,j)}^v$ in order to get more suitable expressions for the mean field theory equations. A close examination of Eqs. (6.12) and (6.13) reveals the symmetry between the expressions for $E_{(i,j)}^h$ and $E_{(i,j)}^v$ terms which can be obtained from each other by interchanging "r" with "c" and "P" with "Q". Hence, algebraic simplifications will only be discussed for the $E_{(i,j)}^v$ term. Similar step can be followed for the $E_{(i,j)}^h$ term.

We introduce the following notation for the sake of simplification of the communication cost terms:

$$F_{i,k}^c = \sum_{l=1}^k v_{il}^c \quad L_{i,k}^c = \sum_{l=k}^Q v_{il}^c, \quad F_{i,k}^r = \sum_{l=1}^k v_{il}^r \quad L_{i,k}^r = \sum_{l=k}^P v_{il}^r \quad (6.14)$$

Here, $F_{i,k}^c$ and $L_{i,k}^c$ denote the probabilities that task i is mapped to one of the processor in the first k columns (i.e., columns $1, 2, 3, \dots, k$) and the last $Q-k+1$ columns (i.e., columns $k, k+1, \dots, Q$), respectively. Similarly, $F_{i,k}^r$ and $L_{i,k}^r$ denote the probabilities that task i is mapped to one of the processors in the first k rows and the last $P-k+1$ rows, respectively. Using this notation and thru some algebraic manipulations the expression for $E_{(i,j)}^h$ simplifies as :

$$\begin{aligned} E_{(i,j)}^h &= e_{ij} \left\{ \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q (l-k) v_{ik}^c v_{jl}^c + \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q (l-k) v_{jk}^c v_{il}^c \right\} \\ &= e_{ij} \left\{ \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q \sum_{m=l}^Q v_{ik}^c v_{jm}^c + \sum_{k=1}^{Q-1} \sum_{l=k+1}^Q \sum_{m=l}^Q v_{jk}^c v_{im}^c \right\} \\ &= e_{ij} \left\{ \sum_{k=1}^{Q-1} \sum_{l=1}^k \sum_{m=k+1}^Q v_{il}^c v_{jm}^c + \sum_{k=1}^{Q-1} \sum_{l=1}^k \sum_{m=k+1}^Q v_{jl}^c v_{im}^c \right\} \\ &= e_{ij} \left\{ \sum_{k=1}^{Q-1} \sum_{l=1}^k v_{il}^c \sum_{m=k+1}^Q v_{jm}^c + \sum_{k=1}^{Q-1} \sum_{l=1}^k v_{jl}^c \sum_{m=k+1}^Q v_{im}^c \right\} \\ &= e_{ij} \sum_{k=1}^{Q-1} F_{i,k}^c L_{j,k+1}^c + e_{ij} \sum_{k=1}^{Q-1} F_{j,k}^c L_{i,k+1}^c \\ &= e_{ij} \sum_{k=1}^{Q-1} (F_{i,k}^c L_{j,k+1}^c + F_{j,k}^c L_{i,k+1}^c) \end{aligned} \quad (6.15)$$

Similarly, the expression for $E_{(i,j)}^v$ simplifies to

$$E_{(i,j)}^v = e_{ij} \sum_{k=1}^{P-1} (F_{i,k}^r L_{j,k+1}^r + F_{j,k}^r L_{i,k+1}^r) \quad (6.16)$$

We formulate the energy term corresponding to the imbalance cost using the same inner product approach adopted in the general formulation (Eq. (6.5)) as follows:

$$\begin{aligned} E^B &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \mathcal{P}(\text{task } i \text{ and } j \text{ are mapped to the same processor}) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{p=1}^P \sum_{q=1}^Q \mathcal{P}(\text{task } i \text{ is mapped to the processor } pq) \\ &\quad \mathcal{P}(\text{task } j \text{ is mapped to the processor } pq) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{p=1}^P \sum_{q=1}^Q v_{ip}^r v_{iq}^c v_{jp}^r v_{jq}^c \end{aligned} \quad (6.17)$$

Total energy term can be defined in terms of the communication cost terms and the imbalance cost term as

$$E(\mathbf{V}^r, \mathbf{V}^c) = E^h(\mathbf{V}^c) + E^v(\mathbf{V}^r) + \beta E^B(\mathbf{V}^r, \mathbf{V}^c) \quad (6.18)$$

Here, $\mathbf{V}^r = [\mathbf{V}_1^r, \dots, \mathbf{V}_p^r, \dots, \mathbf{V}_P^r]^t$ and $\mathbf{V}^c = [\mathbf{V}_1^c, \dots, \mathbf{V}_q^c, \dots, \mathbf{V}_Q^c]^t$ denote the row and column spin-average matrices consisting of N , P and Q dimensional vectors as their rows, respectively.

Derivation of the Mean Field Theory Equation

The expected values \mathbf{V}_i^r and \mathbf{V}_i^c of each row and column spins \mathbf{S}_i^r and \mathbf{S}_i^c are iteratively updated using the Boltzmann distribution as

$$(a) \quad v_{ip}^r = \frac{e^{\phi_{ip}/T^r}}{\sum_{k=1}^P e^{\phi_{ik}/T^r}}, \quad (b) \quad v_{iq}^c = \frac{e^{\phi_{iq}/T^c}}{\sum_{k=1}^P e^{\phi_{ik}/T^c}}, \quad (6.19)$$

for $p = 1, 2, \dots, P$ and $q = 1, 2, \dots, Q$, respectively. Here, T^r and T^c denote the temperature parameters used for annealing the row and column spin updates respectively. Recall that, the number of states of the row and column spins are different (P and Q for row and column spins, respectively) in the proposed encoding. As the convergence time and the temperature parameter of the system depends on the number of states of the spins we interpret the row and column spins as different system, i.e., the temperature parameter of the row and column spins are different. Note that, Eqs. (6.19.a) and (6.19.b) handle the constraints given in Eq. (6.10) thus enforcing each row and column Potts spins \mathbf{S}_i^r and \mathbf{S}_i^c to be in one of the P and Q states when they converge. In the proposed MFA formulation, row and column spins are updated in an alternative manner, i.e., each row spin update is followed with a column spin update and vice versa. MFA iterations in which row and column spins are updated will be referred here as row and column iterations, respectively.

In the proposed formulation, row and column mean field vectors Φ_i^r and Φ_i^c are to be computed in row and column iterations, respectively. Each element ϕ_{ip}^r and ϕ_{iq}^c of the row and column mean field vectors $\Phi_i^r = [\phi_{i1}^r, \dots, \phi_{ip}^r, \dots, \phi_{iP}^r]^t$ and $\Phi_i^c = [\phi_{i1}^c, \dots, \phi_{iq}^c, \dots, \phi_{iQ}^c]^t$ experienced by row and column Potts spins i denote the decrease in the energy function by assigning \mathbf{S}_i^r to \mathbf{e}_p and \mathbf{S}_i^c to \mathbf{e}_q , respectively. Hence, $-\phi_{ip}^r$ ($-\phi_{iq}^c$) may be interpreted as the decrease in the overall solution quality by mapping task i to row p (column q). In other words, $-\phi_{ip}^r$ ($-\phi_{iq}^c$) corresponds to the increase in the energy function

by mapping task i to row p (column q). Then, in Eq. (6.19.a) (Eq. (6.19.b)), v_{ip}^r (v_{iq}^c) is updated such that the probability of mapping task i to row p (column q) increases with increasing mean field value ϕ_{ip}^r (ϕ_{iq}^c). Using the simplified expressions for the proposed energy function in Eqs. (6.15), (6.16) and (6.17)

$$\begin{aligned}\phi_{ip}^r &= -\frac{\partial H(\mathbf{V}^r, \mathbf{V}^c)}{\partial v_{ip}^r} = \phi_{ip}^{r(C)} + \beta^r \phi_{ip}^{r(B)} \\ &= -\sum_{j \in \text{Adj}(i)} e_{ij} Z_{jp}^r - \beta^r w_i \sum_{j=1, j \neq i}^N w_j v_{jp}^r \sum_{q=1}^Q v_{iq}^c v_{jq}^c\end{aligned}\quad (6.20)$$

$$\begin{aligned}\phi_{iq}^c &= -\frac{\partial H(\mathbf{V}^r, \mathbf{V}^c)}{\partial v_{iq}^c} = \phi_{iq}^{c(C)} + \beta^c \phi_{iq}^{c(B)} \\ &= -\sum_{j \in \text{Adj}(i)} e_{ij} Z_{jq}^c - \beta^c w_i \sum_{j=1, j \neq i}^N w_j v_{jq}^c \sum_{p=1}^P v_{ip}^r v_{jp}^r\end{aligned}\quad (6.21)$$

$$\text{where } Z_{jp}^r = \sum_{k=1}^{p-1} F_{jk}^r + \sum_{k=p+1}^P L_{jk}^r \quad \text{and} \quad Z_{jq}^c = \sum_{k=1}^{q-1} F_{jk}^c + \sum_{k=q+1}^Q L_{jk}^c$$

As seen in Eqs.(6.20) and (6.21), different balance parameters β^r and β^c are used in the mean field computations of row and column iterations since row and column spins are interpreted as different system. Figure 6.1 illustrate the MFA algorithm proposed for the mapping problem for mesh-connected architectures. Note that, each iteration of the inner *while-loop* (step 3.1) involves one row and one column iteration. Also note that the computation of the energy differences ΔE^r and ΔE^c necessitates computing E in Eq. (6.18) twice at each iteration of the inner *while-loop* which drastically increases the complexity of a MFA iteration. Here, ΔE^r and ΔE^c represent the energy differences due to the row and column spin updates, respectively. As is seen at Step 3.1.5 we use the efficient energy difference computation scheme which we have proposed for the general MFA formulation [6].

An Efficient Implementation Scheme

As mentioned earlier, the proposed MFA algorithm is an iterative process. The complexity of a single MFA iteration is due mainly to the mean field computations. As is seen in Eqs. (6.20) and (6.21), calculation of mean field values is computationally very intensive. In this section, we propose an efficient implementation scheme which reduces the complexity of mean field computations,

-
1. Get the initial temperatures T_0^r, T_0^c , and set $T^r = T_0^r, T^c = T_0^c$
 2. Initialize the spin averages $\mathbf{V}^r = [v_{11}^r, \dots, v_{ik}^r, \dots, v_{NP}^r]$
and $\mathbf{V}^c = [v_{11}^c, \dots, v_{ik}^c, \dots, v_{NQ}^c]$
 3. WHILE temperature T^r and T^c is in the cooling range DO
 - 3.1 WHILE E^r and E^c are decreasing DO
 - 3.1.1 Select tasks i and j at random for horizontal and vertical spins, respectively.
 - 3.1.2 Compute mean field vectors Φ_i^r and Φ_j^c experienced by row and column Potts spins i and j .

$$\phi_{ip}^r = -\sum_{h \in Adj(i)} e_{ih} Z_{hp}^r - \beta^r w_i \sum_{h=1, h \neq i}^N w_h v_{hp}^r \sum_{q=1}^Q v_{iq}^c v_{hq}^c$$

$$\phi_{jq}^c = -\sum_{h \in Adj(j)} e_{jh} Z_{hq}^c - \beta^c w_j \sum_{h=1, h \neq j}^N w_h v_{hq}^c \sum_{p=1}^P v_{jp}^r v_{hp}^r$$
 - 3.1.3 Compute the summations $\sum_{k=1}^P e^{\phi_{ik}^r/T^r}$ and $\sum_{k=1}^Q e^{\phi_{jk}^c/T^c}$
 - 3.1.4 Compute row and column spin-average vectors \mathbf{V}_i^r and \mathbf{V}_j^c

$$v_{ip}^{r(new)} = e^{\phi_{ip}^r/T^r} / \sum_{k=1}^P e^{\phi_{ik}^r/T^r}$$

$$v_{jq}^{c(new)} = e^{\phi_{jq}^c/T^c} / \sum_{k=1}^Q e^{\phi_{jk}^c/T^c}$$
 - 3.1.5 Compute the energy changes $\Delta E^r = \phi_{ip}^r \Delta v_{ip}^r$ and $\Delta E^c = \phi_{jq}^c \Delta v_{jq}^c$
 - 3.1.6 Update row and column spin-average vectors \mathbf{V}_i^r and \mathbf{V}_j^c

$$v_{ip}^r = v_{ip}^{r(new)} \quad \text{and} \quad v_{jq}^c = v_{jq}^{c(new)}$$
 - 3.2 $T^r = \alpha \times T^r$ and $T^c = \alpha \times T^c$
-

Figure 6.1. The proposed efficient MFA algorithm for the mapping problem for mesh-connected Architectures.

and hence the complexity of the MFA iteration, by asymptotical factors. Mean field theory equations given in Section 6.2.1 reveals the symmetry between the mean field vector computations in row and column iterations. Hence, the proposed implementation scheme will only be discussed for computing the mean field vector $\Phi_i^r = [\phi_{i1}^r, \dots, \phi_{ip}^r, \dots, \phi_{iP}^r]^t$ in row iterations. Similar discussion applies to the computation of the $\Phi_i^c = [\phi_{i1}^c, \dots, \phi_{iq}^c, \dots, \phi_{iQ}^c]^t$ vector in column iterations.

Assume that row Potts spin i is selected at random in a row iteration for updating its expected value vector V_i^r . We will first discuss the mean field computations corresponding to the vertical communication cost. As is seen in Eq. (6.20), these computations require the construction of the $Z_j^r = [Z_{j1}^r, \dots, Z_{jp}^r, \dots, Z_{jP}^r]^t$ vector for each vertex j adjacent to i in TIG. The computation of an individual Z_j^r vector necessitates the construction of $F_j^r = [F_{j1}^r, \dots, F_{jp}^r, \dots, F_{jP}^r]^t$ and $L_j^r = [L_{j1}^r, \dots, L_{jp}^r, \dots, L_{jP}^r]^t$ vectors. These two vectors can be constructed in $\Theta(P)$ time using the recursive equation

$$F_{jk}^r = F_{j,k-1}^r + v_{jk}^r, \quad \text{for } k = 2, 3, \dots, P \quad (6.22)$$

$$\text{where } F_{j1}^r = v_{j1}^r$$

$$L_{jk}^r = L_{j,k-1}^r + v_{jk}^r, \quad \text{for } k = P-1, P-2, \dots, 1 \quad (6.23)$$

$$\text{where } L_{jP}^r = v_{jP}^r$$

The computation of an individual Z_{jp}^r value takes $\Theta(P)$ time. Hence, the complexity of computing an individual Z_j^r vector becomes $\Theta(P^2)$. However, in the proposed scheme the elements of the Z_j^r vector are computed in only $\Theta(P)$ time by exploiting the recursive equation

$$Z_{jq}^r = Z_{j,q-1}^r - L_{jq}^r + F_{j,q-1}^r \quad \text{for } k = 1, 2, \dots, P \quad (6.24)$$

$$\text{where } Z_{j1}^r = \sum_{l=2}^P L_{jl}^r$$

Hence, the complexity of mean field computations corresponding to the vertical communication costs term is $\Theta(d_i P)$ in a row iteration since the first summation term in Eq. (6.20) requires the computation and weighted addition of d_i such Z_j^r vectors. Here, d_i denotes the degree of vertex i in the TIG. Similarly, the complexity of mean field computations corresponding to the horizontal communication cost term is $\Theta(d_i Q)$ when column spin i is selected at random in a column iteration.

As is seen in Eq. (6.20), the complexity of computing an individual mean field value corresponding to the imbalance term is $\Theta(NQ)$. Since P such values

are computed in a row iteration, the total complexity of mean field computations corresponding to the imbalance cost term becomes $\Theta(NPQ)$. However, the complexity of these computations can be asymptotically reduced as follows. The second summation term in Eq. (6.20) can be re-written by interchanging the order of summations as

$$\begin{aligned} w_i \sum_{j=1, j \neq i}^N w_j v_{jp}^r \sum_{q=1}^Q v_{iq}^c v_{jq}^c &= w_i \sum_{q=1}^Q v_{iq}^c \sum_{j=1, j \neq i}^N w_j v_{jp}^r v_{jq}^c \\ &= w_i \sum_{q=1}^Q v_{iq}^c (W_{pq} - w_i v_{ip}^r v_{iq}^c) \end{aligned} \quad (6.25)$$

$$\text{where } W_{pq} = \sum_{j=1}^N w_j v_{jp}^r v_{jq}^c \quad (6.26)$$

Here, W_{pq} denotes the total computational load of processor pq for the current row and column spin values. In Eq. (6.26), $W_{pq} - w_i v_{ip}^r v_{iq}^c$ denotes the weight of processor pq excluding task i . Hence, Eq. (6.26) represents the increase in the imbalance cost term if task i is assigned to row p (i.e., v_{ip}^r is set to 1). In the proposed implementation scheme, we maintain a P by Q processor weight matrix \mathbf{W} consisting of W_{pq} values. The entries of this matrix are computed using Eq. (6.26) only at the beginning of the algorithm. Then, while updating the expected value vector \mathbf{V}_i^r of an individual Potts spin i , the \mathbf{W} matrix is updated in $\Theta(PQ)$ time using

$$W_{pq}^{(new)} = W_{pq}^{(old)} + w_i v_{iq}^c (v_{ip}^{r(new)} - v_{ip}^{r(old)})$$

for $p = 1, 2, \dots, P$ and $q = 1, 2, \dots, Q$. Hence, computing Eq. (6.26) for each ϕ_{ip}^r value takes $\Theta(Q)$ time. Since, P such values are to be computed to construct the mean field vector, the total complexity of mean field computations corresponding to the imbalance cost term reduces $\Theta(PQ)$ in a row iteration.

It should be noted here that, column iterations also use and update the same weight matrix \mathbf{W} as is used and maintained in row iterations. The complexity of mean field computations corresponding to the imbalance cost term is also $\Theta(QP)$ in column iterations. Thus, the proposed scheme reduces the overall complexity of mean field computations to $\Theta(d_{avg}P + PQ)$ and $\Theta(d_{avg}Q + PQ)$ in row and column iteration, respectively. Here, d_{avg} denotes the average vertex degree in TIG. After computing the mean field vectors Φ_i^r and Φ_j^c , expected value vectors \mathbf{V}_i^r and \mathbf{V}_j^c of row and column Potts spin i and j can be updated using Eq. (6.19.a) and Eq. (6.19.b) in $\Theta(P)$ and $\Theta(Q)$ times, in a row and column iteration, respectively. The complexities of computing the energy

difference ΔE^r and ΔE^c as shown at step 3.1.5 of Fig. 6.1 are $\Theta(P)$ and $\Theta(Q)$ times, in a row and column iteration, respectively.

Therefore, the proposed implementation scheme reduces the complexity of an individual row and column iteration to $\Theta(d_{avg}P + PQ)$ and $\Theta(d_{avg}Q + PQ)$, respectively. Note that, a row and a column iteration pair corresponds to a single iteration of the general MFA formulation discussed in Section 6.1. Hence the proposed MFA scheme asymptotically reduces the complexity of a single MFA iteration from $\Theta(d_{avg}PQ + (PQ)^2)$ of the general MFA formulation to $\Theta(d_{avg}(P+Q)+PQ)$ for a P by Q mesh. For a square mesh with K processors, this corresponds to an asymptotical complexity reduction from $\Theta(d_{avg}K + K^2)$ to $\Theta(d_{avg}\sqrt{K} + K)$.

6.2.2 MFA Formulation For Hypercube Architecture

Consider M dimensional hypercube, encoding in the general MFA formulation summarized in Section 6.1 needs $N \times K$ variables for problem representation. Here, N is the number of task and $M = \log(K)$. In this section, we propose a new MFA formulation for hypercube type multicomputers which necessitates $N \times \log(K)$ variables for problem representation. For sake of simplicity, some definition about hypercube are given below. The communication distance between any two processors is equal to *Hamming distance* between those two processors. The *Hamming Distance* between two processors in hypercube is defined as the number of different bits between those two processor id's (binary representation of processor ids). A *dimension i* refers to the communication links between the processors whose processors ids differs on the i th bit. A M dimensional hypercube can be divided into two $(M - 1)$ dimensional subcube along the any dimension. Therefore, M dimensional hypercube can be divided into two $(M - 1)$ subcube in M different ways (dimension). We define two $(M - 1)$ dimensional subcubes H^i and \bar{H}^i which is constructed by dividing M dimensional hypercube along the i th dimension. Subcube H^i contains the processor whose i th bit of ids is 1 and subcube \bar{H}^i contains the processors whose i th bit is 0. In Figure 6.2, the 3-dimensional hypercube is divided into two 2-dimensional subcubes in 3 different ways. In our new efficient formulation, each task is assigned to subcubes instead of processors.

In hypercube topologies, using Ising model is more suitable than Potts model, because in Ising model spins can be in one of the two states represented

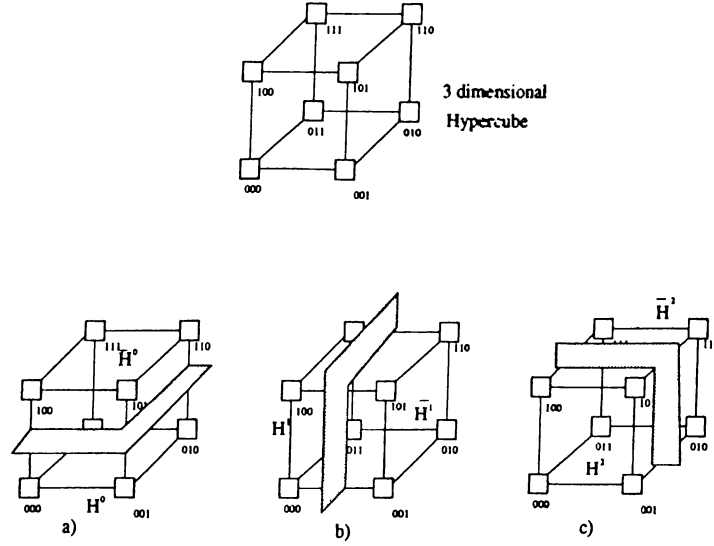


Figure 6.2. Three different ways for dividing 3-dimensional hypercube to 2-dimensional subcubes

by 0 and 1. So, for each $M - 1$ dimensional subcube H^m of the M dimensional hypercube, one Ising spin is used for encoding. To encode the configuration space of the mapping problem, one Ising spin is assigned to each $M - 1$ dimensional subcube of the hypercube. Totally M Ising spin is represented for each task i . Here M is the number of dimension of the hypercube and if there is K processor in hypercube, then $M = \log(K)$.

There will be a total of $|N| \times \log(K)$ Ising spins in the system for encoding the configuration space of the problem. Note that, this encoding constructs the one-to-one mapping between the configuration space of the problem domain and the spin domain. This encoding is much more efficient than the general MFA encoding which requires $|N| \times K$ spins for encoding.

The spin which is assigned to task i and represented to subcube H^i of the hypercube is labeled as s_i^m . If a s_i^m is 1, we say that the corresponding task is mapped to one of the processors the H_m subcube.

The average $v_i^m = \langle s_i^m \rangle$ of each spin. s_i^m is computed and iteratively updated until the system stabilizes at some fixed point. We define

$$v_i^m = \langle s_i^m \rangle \quad \text{where} \quad m = 1, 2, \dots, \log(K)$$

Here $s_i^m \in \{0, 1\}$, whereas $v_i^m \in [0, 1]$, In order to construct an energy function,

it is helpful to associate the following meaning to v_i^m values.

$$v_i^m = \mathcal{P}\{\text{task } i \text{ is mapped to one of the processors in subcube } H^m\}$$

For simplicity, the energy computation is divided to two part, interconnection communication energy term (E_{com}) and imbalance energy term (E_{bal}).

$$E = E_{com} + r \times E_{bal}$$

We derive the interconnection communication energy function for mapping problem as follows.

$$\begin{aligned} E_{com} &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N e_{ij} \sum_{l=1}^{\log(K)} \mathcal{P}\{\text{task } i \text{ is mapped to one of the processor in } H^l\} \times \\ &\quad \mathcal{P}\{\text{task } j \text{ is mapped to one of the processors in } \bar{H}^l\} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N e_{ij} \sum_{l=1}^{\log(K)} s_i^l \times (1 - s_j^l) \end{aligned} \quad (6.27)$$

We consider the load-imbalance term for each processors so we formulate the energy term correspond the imbalance cost as

$$\begin{aligned} E_{bal} &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{p=1}^K \mathcal{P}\{\text{task } i \text{ is mapped to processor } p\} \times \\ &\quad \mathcal{P}\{\text{task } j \text{ is mapped to processor } p\} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{p=1}^K \mathcal{S}_i^p \mathcal{S}_j^p \end{aligned} \quad (6.29)$$

Here, \mathcal{S}_i^p is the probability of task i is mapped to processor p . For example, we have 4-dimensional hypercube and the probability of task i mapped to processor 9 is $\mathcal{S}_i^9 = [s_i^4 s_i^3 s_i^2 s_i^1] = (s_i^4 \times (1 - s_i^3) \times (1 - s_i^2) \times s_i^1)$ we define \mathcal{S}_i^p as

$$\mathcal{S}_i^p = \prod_{l=1}^{\log(p)} z_i^l \quad \text{where} \quad z_i^l = m s_i^l + \bar{m}(1 - s_i^l) \quad (6.30)$$

Here z_i^l is s_i^l or $(1 - s_i^l)$ according to the binary representation of the processor number p . In equation (6.30), m is 1 or 0 if the l -th bit of the processor number is 1 or 0. Total energy term can be defined in terms of communication cost term and the imbalance term as

$$\begin{aligned} E &= E_{com} + r \times E_{bal} \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N e_{ij} \sum_{l=1}^{\log(K)} s_i^l \times (1 - s_j^l) + \\ &\quad r \times \frac{1}{2} \sum_{i=1}^N \sum_{j \neq i}^N w_i w_j \sum_{p=1}^K \mathcal{S}_i^p \mathcal{S}_j^p \end{aligned} \quad (6.31)$$

In MFA algorithm, the expected values v_i^m of each spin s_i^m are iteratively updated using Boltzmann distribution as

$$v_i^m = \frac{1}{1 + e^{-\phi_i^m/T}} \quad (6.32)$$

Each ϕ_i^m denotes the decrease in the energy function. Hence for the formulation of mapping problem for hypercube $-\phi_i^m$ may be interpreted as the decrease in the overall solution quality by assigning task i to one of the processors in subcube H^m . In this work the mean field values are computed as

$$\phi_i^m = \phi_{com,i}^m + r \times \phi_{bal,i}^m$$

The mean field values coming from the communication energy term is calculated as

$$\phi_{com,i}^m = \frac{\partial E_{com}}{\partial s_i^m} = - \sum_{j \in Adj(i)} e_{ij} (v_j^m - \frac{1}{2}) \quad (6.33)$$

Here if $\phi_{com,i}^m$ is positive then v_i^m is attracted to 1. This means that probability of task i is mapped to one of the processor whose m -th bit is 1. Also, if $\phi_{com,i}^m$ is negative then v_i^m is attracted to 0. This means that probability of task i is not mapped to one of the processor whose m -th bit of binary number is 1. The computation of the mean field value for communication cost takes $O(d_{avg})$ where d_{avg} is the average vertex degree of TIG.

Second Term of the mean field value is the imbalance energy term is calculated as

$$\begin{aligned} \phi_{bal,i}^m &= \frac{\partial E_{bal}}{\partial s_i^m} = -\frac{1}{2} \sum_{j=1}^N \sum_{p=1}^K \alpha w_i w_j \prod_{k=1, k \neq m}^{log(K)} z_i^k \prod_{l=1}^{log(K)} z_j^l \\ &= -\frac{1}{2} w_i \sum_{p=1}^K \alpha \prod_{k=1, k \neq m}^{log(K)} z_i^k \sum_{j=1, j \neq i}^N w_j \prod_{l=1}^{log(K)} z_j^l \end{aligned} \quad (6.34)$$

Here α is 1 or -1 according to m -th bit of the processor p . To simplify the equation (6.34), the product term is substituted by S_i^m in equation (6.30).

$$\phi_{b,i}^m = -\frac{1}{2} w_i \sum_{p=1}^K \alpha \frac{S_i^p}{s_i^m} \sum_{j=1, j \neq i}^N w_j S_j^p \quad (6.35)$$

As seen equation (6.35), the complexity of computing an individual mean field value corresponding the imbalance cost is $O(|N| \times K)$. However, the complexity

of the computation can be asymptotically reduced as follows.

$$\phi_{b,i}^m = -\frac{1}{2} \sum_{p=1}^{\log(K)} \alpha(\mathcal{S}_i^p/s_i^m)(\mathcal{W}^p - \mathcal{S}_i^p) \quad (6.36)$$

$$\text{where } \mathcal{W}^p = \sum_{j=1}^N w_j \mathcal{S}_j^p \quad (6.37)$$

Here, \mathcal{W}^p denotes the weight of the processor p for current spin values. The parenthesis term inside the summation (6.36) denotes the weight of processor p excluding the task i . Hence (6.36) represent the increase in imbalance cost term, if task i is assigned to processor p . The entries of the \mathcal{W} vectors are computed using (6.37) at the beginning of the algorithm. Then, while updating the expected value of individual Ising spin i , the \mathcal{W} vector is updated in $O(K)$ by using iterative properties of equation (6.37). If the s_i^m is updated in MFA iteration then the \mathcal{W} vector is updated like as

$$\mathcal{W}_{new}^p = \mathcal{W}_{old}^p + \mathcal{S}_i^{p(new)} - \mathcal{S}_i^{p(old)} \quad \text{where } \mathcal{S}_i^{p(new)} = \frac{\mathcal{S}_i^{p(old)}}{s_i^{m(old)}} \times s_i^{m(new)} \quad (6.38)$$

As the \mathcal{S}_i^p value is updated in $O(1)$ times, updating the \mathcal{W} vector takes $O(K)$ times. Therefore total computation of mean field value for imbalance cost term ($\phi_{b,i}^m$) takes $O(K)$ times.

In Figure 6.3, another method are given for calculating the mean field value for imbalance cost term which takes also $O(K)$.

If we add the mean field values from communication cost term (6.33) and imbalance term (6.36), the mean field value for given spin i and subcube H^m is

$$\phi_i^m = - \sum_{j \in Adj(i)} e_{ij} (v_j^m - \frac{1}{2}) - \frac{1}{2} \sum_{p=1}^{\log(K)} \alpha(\mathcal{S}_i^p/s_i^m)(\mathcal{W}^p - \mathcal{S}_i^p) \quad (6.39)$$

As seen in (6.39), total computation of the mean field value for given spin i and dimension m is $O(d_{avg} + K)$. Steps of the MFA algorithm for hypercube topologies is very similar to the MFA algorithm for mesh. In this MFA algorithm one spin is selected randomly for each dimension. Therefore one MFA iteration requires $\log(K)$ mean field value computation. So complexity of the one MFA iteration is $O(d_{avg} \times \log K + K \times \log K)$. Instead of $O(d_{avg} \times K + K^2)$ in the traditional MFA algorithm.

```

sum = 0;
for k=0 to (p/2k+1)-1 do
  for l=0 to 2k - 1 do
    p = i × 2k+1 + l;
    q = p + 2k

    Wp = Wp - wiSip
    Wq = Wq - wiSiq
    sum = sum + Sip(Wq - Wp
  endfor
endfor
φim = -wi × (sum/sim)

```

Figure 6.3. The Mean field value calculation of given spin i of subcube H^m

6.3 Performance Evaluation

This section presents the performance evaluation of the efficient MFA formulation proposed for the mapping problem for mesh-connected architectures in comparison with the well known mapping heuristics: simulated annealing (SA), Kernighan-Lin (KL) and the general MFA formulation. Each algorithm is tested using randomly generated mapping problem instances for mesh-connected architectures. The following paragraphs briefly present the implementation details of these algorithms.

The MFA algorithm proposed for the mapping problem for mesh topology is implemented efficiently as described in Section 6.2.1. At the very beginning the of the algorithm row and column spin averages are initialized to $1/P$ and $1/Q$ plus a random disturbance term, so that the initial spin averages are uniformly distributed in the range

$$0.9 \times \frac{1}{P} \leq v_{ip}^{r(initial)} \leq 1.1 \times \frac{1}{P} \quad \text{for } i = 1, 2, \dots, N, \quad p = 1, 2, \dots, P$$

$$0.9 \times \frac{1}{Q} \leq v_{iq}^{c(initial)} \leq 1.1 \times \frac{1}{Q} \quad \text{for } i = 1, 2, \dots, N, \quad q = 1, 2, \dots, Q$$

respectively. Note that $\lim_{T^r \rightarrow \infty} v_{ip}^r = 1/P$ and $\lim_{T^c \rightarrow \infty} v_{iq}^c = 1/Q$. The initial temperatures and balance parameters used in the mean field computation of the row and column iterations are estimated using these initial random spin average values. Recall that, in the mean field computations (Eqs. (6.20))

and (6.21) of row and column iterations, the parameters β^r and β^c determine a balance between the terms $\phi_{ip}^{r(C)}$ and $\phi_{ip}^{r(B)}$ and $\phi_{iq}^{c(C)}$ and $\phi_{iq}^{c(B)}$, respectively. We compute the row spin averages $\langle \phi_{ip}^{r(C)} \rangle = (\sum_{i=1}^N \sum_{p=1}^P \phi_{ip}^{r(C)})/NP$ and $\langle \phi_{ip}^{r(B)} \rangle = (\sum_{i=1}^N \sum_{p=1}^P \phi_{ip}^{r(B)})/NP$ using the initial v_{ip}^r values. Column spin averages $\langle \phi_{iq}^{c(C)} \rangle$ and $\langle \phi_{iq}^{c(B)} \rangle$ are computed similarly using the initial v_{iq}^c values. Then, balance parameters are computed as $\beta^r = C_B \langle \phi_{ip}^{r(C)} \rangle / \langle \phi_{ip}^{r(B)} \rangle$ and $\beta^c = C_B \langle \phi_{iq}^{c(C)} \rangle / \langle \phi_{iq}^{c(B)} \rangle$, where C_B is chosen as 5.6. Our experiments show that computing β^r and β^c using this method is sufficient for obtaining balanced partitions.

Selection of initial temperature parameters T_0^r and T_0^c is crucial for obtaining good quality solutions. In previous applications of MFA [18, 22], it is experimentally observed that spin averages tend to converge at a critical temperature. Although there are some methods proposed for the estimation of critical temperature, we prefer an experimental way for computing T_0^r and T_0^c which is easy to implement and successful as the results of experiments indicate. After the balance parameters β^r and β^c are fixed, average row and column mean fields are computed as $\langle \phi_{ip}^r \rangle = \langle \phi_{ip}^{r(C)} \rangle + \beta^r \langle \phi_{ip}^{r(B)} \rangle$ and $\langle \phi_{iq}^c \rangle = \langle \phi_{iq}^{c(C)} \rangle + \beta^c \langle \phi_{iq}^{c(B)} \rangle$. Then T_0^r and T_0^c are computed using $T_0^r = C_T \langle \phi_{ip}^r \rangle / P$ and $T_0^c = C_T \langle \phi_{iq}^c \rangle / Q$ where C_T is chosen as 20. Note that, both T_0^r and T_0^c are inversely proportional to the dimensions of the row and column Potts spins, respectively, which is also observed for the critical temperature formulations presented in other MFA implementations [18, 26].

The same cooling schedule is adopted for row and column iterations as follows. At each temperature, row and column iterations proceed in an alternative manner for randomly selected unconverged row and column spin updates until $\Delta E^r < \epsilon$ and $\Delta E^c < \epsilon$ for M consecutive iterations respectively where $M = N$ initially and $\epsilon = 0.05$. Average spin values are tested for convergence after each update. If one of the v_{ik} terms of a row or column spin average vector is detected to be greater than 0.95, that spin is assumed to converge to state k . The cooling process is realized in two phases, slow cooling followed by fast cooling, similar to the cooling schedules used for SA [22]. In the slow cooling phase, row and column temperatures are decreased using $\alpha = 0.9$ until $T < T_0/1.5$ for both row and column iterations. Then in the fast cooling phase, M is set to $M/4$, α is set to 0.7 and cooling for row and column iterations are continued until 90% of the row and column spins converge, respectively. At the end of this cooling process, the maximum element in each unconverged spin average vector is set to 1 and all other elements in that vector are set to

0. Then, the result is decoded as described in Section 6.2, and the resulting mapping is found. Note that, all parameters used in this implementation are either constants or found automatically. Hence, there is no parameter setting problem for different mapping instances.

The general MFA formulation summarized in Section 6.1 is implemented efficiently as described in [6]. The initialization of spin averages, the selection of the balance parameter β and the initial temperature T_0 are performed as is described for the mesh-specific MFA implementation. The expressions used for these computations can be found by replacing P and Q with $K = P \times Q$ in those expressions described for the mesh-specific MFA implementation. The parameters C_T and C_B are chosen as 0.5. The same cooling schedule described for mesh-specific MFA implementation is used in the implementation of the general MFA formulation.

The two-phase approach is used to apply KL to the mapping problem. KL heuristics is implemented efficiently as described by Fiduccia and Mattheyses (FM) [6] for the clustering phase. The recursive bisection scheme implemented for the first phase recursively partitions the initial TIG into two cluster until $K = P \times Q$ clusters are obtained. Here, K is assumed to be a power of two. In the KLFM heuristic, computational load balance among clusters is maintained implicitly by the algorithm. Vertex moves causing intolerable load imbalance are not considered. The one-to-one mapping heuristics used in the second phase is a variant of the KL heuristics. In this heuristic, communication cost is minimized by performing a sequence of cluster swaps between the processor pairs after an initial random mapping of K clusters [21].

The SA algorithm implemented in this work implicitly achieves the load balance among processors by setting a neighborhood configuration consisting of all configurations which result from moving one task from the processor with maximum load to any other processor. Randomly selected possible moves which decrease the communication costs are realized. Acceptance probabilities of randomly selected moves that increase the communication cost are controlled with a temperature parameter T which is decreased using an automatic annealing schedule [22]. Hence, as the annealing proceeds acceptance probabilities of uphill moves decrease.

Table 6.1. Total communication costs averages normalized with respect to mesh-specific MFA of the solution found by SA,KL,general MFA and mesh-specific MFA for randomly generated mapping problem instances for various mesh size

Problem Size			Average Communication Cost			
TIG		Mesh	KL	SA	MFA	
N	d_{avg}	$P \times Q$			Gen.	Mesh
400	2	4×4	1.20	0.83	1.16	1.00
	2	4×8	2.62	0.76	1.09	1.00
	3	4×4	1.14	1.01	1.13	1.00
	3	4×8	1.96	0.94	1.07	1.00
	4	4×4	1.31	1.03	1.09	1.00
	4	4×8	1.92	0.97	1.08	1.00
800	2	4×8	1.73	0.89	1.10	1.00
	2	8×8	2.61	0.88	1.30	1.00
	3	4×8	2.20	1.13	1.41	1.00
	3	8×8	2.88	1.06	1.00	1.00
	4	4×8	1.65	1.14	1.13	1.00
	4	8×8	2.55	1.17	1.20	1.00
1600	2	8×8	1.61	0.99	0.93	1.00
	2	8×16	2.89	1.05	1.15	1.00
	3	8×8	1.57	0.99	0.96	1.00
	3	8×16	2.47	1.00	1.13	1.00
	4	8×8	2.03	1.17	1.31	1.00
	4	8×16	3.39	0.93	1.26	1.00

6.4 Experimental Results

The mapping heuristics are experimented by mapping randomly generated TIGs and test TIGs onto various size meshes. Random TIGs are generated using the following parameters: number of vertices (N), average vertex degree (d_{avg}), maximum vertex weight (w_{max}) and maximum edge weight (e_{max}). In a random graph $G_{N,p}$ with N vertices, each pair of vertices constitutes an edge with probability p . Since $G_{N,p}$ can have at most $pC(N, 2)$ edges, the sum of the degrees of the vertices of $G_{N,p}$ is equal to $2pC(N, 2)$. Then, the expected average vertex degree of $G_{N,p}$ is $d_{avg} = 2pC(N, 2)/N = p(N - 1)$. Thus, the parameter P is selected as $p = d_{avg}/(N - 1)$ to generate a random TIG with N vertices and expected vertex degree d_{avg} . Then, the edge set is created by flipping a coin with probability p for all $(N(N - 1)/2)$ potential edges. Each vertex or edge is weighted randomly by choosing a number between 1 and w_{max} or 1 and e_{max} , respectively. Nine test TIGs generated with $N = 400, 800, 1600, d_{avg} = 2, 3, 4, w_{max} = 5$ and $e_{max} = 10$ using this random graph generation algorithm. These test TIGs are mapped to $4 \times 4, 4 \times 8, 8 \times 8$ and 8×16 two-dimensional meshes.

Table 6.2. Percent computational load imbalance averages of the solution found by SA, KL, general MFA and mesh-specific MFA for randomly generated mapping problem instances for various mesh size

Problem Size			Average Percent Imbalance			
TIG		Mesh	KL	SA	MFA	
N	d_{avg}	$P \times Q$			Gen.	Mesh
400	2	4×4	9.1	2.1	8.6	7.8
	2	4×8	14.5	6.5	11.1	8.3
	3	4×4	11.4	4.4	8.6	4.5
	3	4×8	15.5	5.5	9.7	8.3
	4	4×4	11.9	4.0	5.1	7.9
	4	4×8	16.1	7.8	12.7	6.3
800	2	4×8	12.0	5.8	16.2	7.8
	2	8×8	16.7	8.4	12.7	8.7
	3	4×8	15.6	3.5	8.7	5.2
	3	8×8	19.7	9.6	16.0	8.2
	4	4×8	16.5	13.8	7.9	14.2
	4	8×8	19.0	6.6	6.2	6.9
1600	2	8×8	13.8	9.3	12.7	8.2
	2	8×16	21.0	9.4	13.9	7.9
	3	8×8	15.3	14.3	16.6	10.3
	3	8×16	19.7	10.9	13.0	11.7
	4	8×8	15.6	9.4	14.9	8.9
	4	8×16	21.9	7.3	11.2	9.4

Table 6.3. Execution time averages of the solution found by SA, KL, general MFA and mesh-specific MFA for randomly generated mapping problem instances for various mesh size

Problem Size			Average Execution Time(sec)			
TIG		Mesh	KL	SA	MFA	
N	d_{avg}	$P \times Q$			Gen.	Mesh
400	2	4×4	1.1	99.4	11.7	2.8
	2	4×8	1.1	99.4	11.7	2.8
	3	4×4	0.9	44.0	3.1	0.9
	3	4×8	1.4	96.4	5.6	1.8
	4	4×4	1.0	48.8	2.7	1.4
	4	4×8	1.5	80.0	9.7	3.5
800	2	4×8	1.7	248.9	15.8	5.3
	2	8×8	3.2	522.8	53.8	6.8
	3	4×8	2.2	256.0	13.0	4.2
	3	8×8	4.4	550.2	44.7	8.6
	4	4×8	2.9	240.2	55.1	8.7
	4	8×8	5.5	545.7	87.6	9.9
1600	2	8×8	5.4	1983.6	230.6	13.5
	2	8×16	15.6	16793.4	1081.5	39.5
	3	8×8	8.9	1826.5	157.2	18.2
	3	8×16	24.1	4946.0	515.0	40.6
	4	8×8	11.3	3095.6	206.2	15.1
	4	8×16	51.0	5345.7	495.4	49.9

Table 6.4. Average performance measures of the solution found by SA, KL, general MFA and mesh-specific MFA for randomly generated mapping problem instances

	KL	SA	MFA	
			Gen.	Mesh.
COMM. COST	2.10	1.00	1.13	1.00
LOAD IMBALANCE	2.01	0.91	1.49	1.00
EXECUTION TIME	0.67	93.20	8.17	1.00

Table 6.1, 6.2, 6.3 illustrates the performance result of the KL, SA, general and mesh-specific MFA heuristics for the generated mapping problem instances. In this table, "Gen" and "Mesh" denote the general and mesh-specific MFA formulations, respectively, discussed in this work. Each algorithm is executed 5 times for each problem instance starting from different, randomly chosen initial configurations. Total communication cost averages of the solutions in Table 6.1 are normalized with respect to the results of the mesh specific MFA heuristic developed in this work. Percent computational load imbalance averages of solutions displayed in Table 6.2 are computed using $100 \times (CL_{max} - CL_{min}) / CL_{avg}$. Here, CL_{max} and CL_{min} denotes the maximum and minimum processor loads and CL_{avg} denote the computational loads of processors under perfect load balance conditions. Execution time averages are measured on a DEC Alpha workstation in seconds for randomly generated mapping problem instances. Table 6.4 is constructed for a better illustration of the overall relative performances of the heuristics. Percent load imbalance averages and execution time averages of the solutions are also normalized with respect to the results of the mesh-specific MFA heuristic. Then, the overall averages of the normalized averages of Table 6.1, 6.2, 6.3 are displayed in Table 6.4.

These four tables confirm the expectation that mesh-specific MFA formulation is significantly faster (8.17 times on the average) than the general MFA formulation while producing solutions with considerably better qualities for randomly generated problem instances. As seen in these tables, the mesh specific MFA heuristic produces significantly better solutions than the KL heuristic whereas the MFA heuristic is slightly slower (only 1.49 times on the average). The qualities of the solutions obtained by the mesh-specific MFA heuristic are comparable with those of the SA heuristic. However, the mesh-specific MFA heuristic is orders of magnitudes faster (93.2 times on the average). Hence, the proposed MFA heuristic approaches the speed performance of the fast KL

Table 6.5. The Benchmark Sparse Matrix data used in experiments

Benchmark	#of Nodes	# of Edges	Min.Deg	Max.Deg	Avg.Deg
DWT-492	492	1332	2	10	5.41
DWT-758	758	2618	3	10	6.91
DWT-1242	1242	4592	1	11	7.39
BCSPWR06	1454	1923	1	12	2.64
BCSPWR09	1723	2394	1	14	2.78
JAGMESH2	1009	2928	3	6	5.80
JAGMESH6	1377	3808	2	6	5.53
JAGMESH7	1138	3156	3	6	5.54
LSHP2233	2233	6552	3	6	5.87
LSHP3466	3466	10215	3	6	5.89

heuristic while approaching the solution quality of the powerful SA heuristic.

Test TIG's correspond to the undirected sparse graphs associated with the symmetric sparse matrices selected from *Harwel Boeing sparse matrix test collection* [12]. Weights of the vertices are assumed to be equal to their degrees. These test TIG's are mapped to 8×8 , 8×16 and 16×16 2D-meshes. The properties of test TIGs are shown in Table 6.5

Table 6.6, 6.7, 6.8 illustrates the performance result of the KL, SA, general and mesh-specific MFA heuristics for the mapping problem instances from test TIGs. Each algorithm is executed 5 times for each problem instance starting from different, randomly chosen initial configurations. Total communication cost averages of the solutions in Table 6.6 are normalized with respect to the results of the mesh specific MFA heuristic developed in this work. Execution time averages are measured on a SUN SPARC 10 workstation. Execution time averages are normalized with respect to those of mesh-specific MFA heuristic in Table 6.8. Table 6.9 is constructed for a better illustration of the overall relative performances of the heuristics. Percent load imbalance averages of the solutions are also normalized with respect to the results of the mesh-specific MFA heuristic. Then, the overall averages of the normalized averages of Table 6.6, 6.7, 6.8 are displayed in Table 2. Tables 6.6, 6.7, 6.8, 6.9 confirm the expectation that mesh-specific MFA formulation is significantly faster (7.26 times on the average) than the general MFA formulation while producing solutions with considerably better qualities for test TIGs. As seen in these tables, the mesh specific MFA heuristic produces significantly better solutions than the KL heuristic whereas the MFA heuristic is slightly slower. The qualities of the solutions obtained by the mesh-specific MFA heuristic are comparable with those of the SA heuristic. However, the mesh-specific MFA heuristic is faster

Table 6.6. Total communication cost averages, normalized with respect to mesh-specific MFA, of the solution found by SA, KL, general MFA and mesh-specific MFA for some benchmark mapping problem instances for various mesh size

Circuit	Par	Com.Cost			
		MFA	SA	GenMFA	KL
DWT-492	16	1.00	0.82	1.39	0.95
	32	1.00	1.11	1.89	1.61
	64	1.00	0.97	1.74	1.98
	128	1.00	1.13	2.52	2.33
	256	1.00	1.10	2.62	1.90
DWT-758	16	1.00	0.83	1.48	0.74
	32	1.00	0.95	1.98	1.17
	64	1.00	0.95	2.02	1.79
	128	1.00	1.10	2.75	2.85
	256	1.00	1.38	4.03	3.34
DWT-1242	16	1.00	0.85	1.18	0.99
	32	1.00	0.95	1.71	1.25
	64	1.00	1.00	2.01	1.42
	128	1.00	1.05	2.62	2.53
	256	1.00	1.08	2.94	2.91
JAGMESH2	16	1.00	0.89	1.12	0.89
	32	1.00	0.93	1.30	0.99
	64	1.00	0.90	2.04	1.91
	128	1.00	1.11	3.35	3.06
	256	1.00	1.19	3.73	3.44
JAGMESH6	16	1.00	0.56	0.92	0.69
	32	1.00	0.87	1.43	1.14
	64	1.00	0.91	1.78	1.23
	128	1.00	1.13	3.59	2.48
	256	1.00	1.08	3.82	3.43
JAGMESH7	16	1.00	0.78	1.12	0.83
	32	1.00	0.86	1.26	1.21
	64	1.00	0.95	1.89	1.40
	128	1.00	1.06	3.25	2.74
	256	1.00	1.20	3.77	3.48
BCSPWR06	16	1.00	0.67	2.14	1.47
	32	1.00	0.98	3.25	2.33
	64	1.00	0.93	2.80	2.18
	128	1.00	1.12	3.35	2.90
	256	1.00	1.23	3.45	3.80
BCSPWR09	16	1.00	0.51	1.36	1.11
	32	1.00	0.89	2.74	1.88
	64	1.00	0.90	2.43	1.87
	128	1.00	1.01	3.13	2.33
	256	1.00	1.80	5.06	4.75
LSHP2233	16	1.00	0.84	1.02	1.09
	32	1.00	0.89	1.29	1.31
	64	1.00	0.81	1.88	1.37
	128	1.00	0.97	3.63	2.20
	256	1.00	1.12	2.68	3.31
LSHP3346	16	1.00	0.65	1.05	0.37
	32	1.00	0.66	1.23	0.43
	64	1.00	0.68	1.91	0.52
	128	1.00	0.68	3.48	0.68
	256	1.00	0.87	2.10	1.07

Table 6.7. Load Imbalanced averages, of the solution found by SA, KL, general MFA and mesh-specific MFA for some benchmark mapping problem instances for various mesh size

Circuit	Par	Load-Bal			
		MFA	SA	GenMFA	KL
DWT-492	16	2.41	2.41	4.34	5.42
	32	3.01	3.61	7.47	7.35
	64	6.10	7.32	8.54	9.76
	128	11.00	15.00	15.50	17.00
	256	19.00	35.00	26.00	28.00
DWT-758	16	1.62	0.92	3.79	6.45
	32	2.45	2.15	5.52	9.45
	64	4.20	5.25	5.68	9.38
	128	7.75	14.37	9.25	12.25
	256	9.00	26.25	15.00	16.50
DWT-1242	16	1.13	0.57	3.55	7.86
	32	1.60	1.48	4.60	8.08
	64	2.66	3.85	6.22	8.88
	128	5.35	5.28	8.17	12.11
	256	9.43	11.43	10.29	16.29
JAGMESH2	16	1.58	0.82	2.51	4.29
	32	0.87	1.64	3.55	5.96
	64	2.64	4.12	5.60	8.13
	128	2.89	6.67	5.56	10.89
	256	6.82	15.91	12.73	18.18
JAGMESH6	16	1.03	0.84	3.95	4.41
	32	1.60	0.84	8.32	6.34
	64	2.10	2.52	7.06	7.39
	128	2.54	4.24	5.25	12.03
	256	7.93	12.07	10.69	13.45
JAGMESH7	16	1.29	0.82	2.89	4.64
	32	1.68	1.27	5.18	6.60
	64	2.86	2.81	6.33	8.06
	128	4.49	7.65	5.92	11.02
	256	9.17	18.75	12.50	13.75
BCSPWR06	16	1.13	0.31	2.92	4.05
	32	2.67	0.63	5.42	5.50
	64	3.33	0.83	8.00	10.54
	128	5.00	1.67	7.67	12.43
	256	8.00	5.00	11.33	17.22
BCSPWR09	16	1.84	0.33	2.31	4.05
	32	2.55	0.67	5.44	5.50
	64	4.19	1.35	7.97	10.54
	128	4.05	2.70	10.81	12.43
	256	4.05	5.56	18.89	17.22
LSHP2233	16	0.88	0.31	1.31	5.16
	32	1.52	0.98	2.44	6.36
	64	2.30	1.23	5.39	8.04
	128	2.45	2.94	3.92	9.41
	256	3.73	7.84	12.07	10.20
LSHP3466	16	0.51	0.31	1.21	4.05
	32	2.02	0.98	1.87	5.50
	64	1.50	1.23	4.48	10.54
	128	1.51	2.94	4.47	12.43
	256	4.18	7.84	12.07	17.22

Table 6.8. Total execution time, normalized with respect to mesh-specific MFA, of the solution found by SA, KL, general MFA and mesh-specific MFA for some bechmark mapping problem instances for various mesh size

Circuit	Par	Execution Time			
		MFA	SA	GenMFA	KL
DWT-492	16	1.00	54.70	3.09	0.24
	32	1.00	16.73	2.78	0.12
	64	1.00	17.56	4.27	0.29
	128	1.00	4.64	1.70	0.33
	256	1.00	3.91	2.45	2.28
DWT-758	16	1.00	63.29	2.48	0.19
	32	1.00	24.00	2.17	0.11
	64	1.00	15.98	3.34	0.15
	128	1.00	5.70	1.63	0.23
	256	1.00	5.39	2.65	1.69
DWT-1242	16	1.00	89.19	6.10	0.18
	32	1.00	27.50	5.01	0.08
	64	1.00	25.33	7.74	0.13
	128	1.00	8.72	2.67	0.19
	256	1.00	7.02	3.79	0.75
JAGMESH2	16	1.00	61.11	8.62	0.12
	32	1.00	24.16	7.69	0.08
	64	1.00	16.43	10.81	0.11
	128	1.00	8.53	4.14	0.24
	256	1.00	8.21	5.27	1.16
JAGMESH6	16	1.00	112.12	10.72	0.16
	32	1.00	45.16	11.93	0.09
	64	1.00	30.02	15.45	0.13
	128	1.00	13.01	6.60	0.18
	256	1.00	10.98	6.25	0.81
JAGMESH7	16	1.00	78.00	7.75	0.15
	32	1.00	32.29	10.98	0.09
	64	1.00	26.58	19.41	0.14
	128	1.00	11.01	4.22	0.20
	256	1.00	9.58	6.77	1.10
BCSPWR06	16	1.00	213.22	2.14	0.30
	32	1.00	66.53	1.74	0.13
	64	1.00	55.05	4.01	0.20
	128	1.00	18.43	4.80	0.26
	256	1.00	14.24	5.88	0.87
BCSPWR09	16	1.00	261.90	3.54	0.24
	32	1.00	76.14	3.81	0.10
	64	1.00	59.62	8.27	0.15
	128	1.00	23.50	6.56	0.20
	256	1.00	32.09	14.88	1.30
LSHP2233	16	1.00	104.60	7.72	0.09
	32	1.00	44.17	10.05	0.06
	64	1.00	34.47	17.28	0.09
	128	1.00	17.48	7.22	0.13
	256	1.00	13.95	2.19	0.57
LSHP3466	16	1.00	53.11	11.11	0.03
	32	1.00	22.63	12.44	0.02
	64	1.00	15.81	13.36	0.02
	128	1.00	8.53	11.62	0.04
	256	1.00	8.48	2.19	0.20

Table 6.9. Average performance measures of the solutions found by SA, KL, general MFA and mesh-specific MFA for mapping problem instances.

	KL	SA	MFA	
			Gen.	Mesh.
Communication Cost	2.55	1.08	2.94	1.00
Load Imbalance	2.34	1.5	1.85	1.00
Execution Time	0.5	19.7	7.26	1.00

Table 6.10. Total communication costs averages normalized with respect to hypercube-specific MFA of the solution found by SA, KL, general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercube size

Problem Size			Average Communication Cost			
TIG		Hypercube	KL	SA	MFA	
N	d_{avg}	K			Gen.	Mesh
400	3	8	1.41	0.96	1.12	1.00
	3	16	2.45	1.02	0.69	1.00
	4	16	2.43	1.32	1.74	1.00
	4	32	1.48	1.21	1.25	1.00
	8	32	1.35	1.18	1.25	1.00
	8	64	1.25	1.18	1.08	1.00
800	3	8	1.39	0.87	1.23	1.00
	3	16	1.47	1.34	1.30	1.00
	4	16	1.73	1.13	1.26	1.00
	4	32	1.83	0.88	0.93	1.00
	8	32	1.55	0.99	1.16	1.00
	8	64	1.42	1.03	1.13	1.00
1600	3	8	1.37	0.92	0.84	1.00
	3	16	0.98	0.74	0.88	1.00
	4	16	0.86	0.74	1.14	1.00
	4	32	1.56	0.87	1.26	1.00
	8	32	1.26	0.89	0.98	1.00
	8	64	1.68	1.14	1.36	1.00

(19.7 times on the average). Hence, the proposed MFA heuristic approaches the speed performance of the fast KL heuristic while approaching the solution quality of the powerful SA heuristic.

Table 6.10, 6.11, 6.12 illustrates the performance result of the KL, SA, general and hypercube-specific MFA heuristics for the generated mapping problem instances. In this table, "Gen" and "Hypercube" denote the general and hypercube-specific MFA formulations, respectively. Each algorithm is executed 10 times for each problem instance starting from different, randomly chosen initial configurations. Total communication cost averages of the solutions in Table 6.10 are normalized with respect to the results of the mesh specific MFA heuristic developed in this work. Percent computational load imbalance averages of solutions displayed in Table 6.2 are computed using

Table 6.11. Percent computational load imbalance averages of the solution found by SA,KL,general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercube size

Problem Size			Average Percent Imbalance			
TIG		Hypercube	KL	SA	MFA	
N	d_{avg}	$P \times Q$			Gen.	Mesh
400	3	8	12.22	7.50	9.17	2.78
	3	16	15.56	8.33	18.46	6.67
	4	16	14.44	9.33	16.43	10.05
	4	32	21.43	15.29	23.33	23.81
	8	32	15.48	12.60	30.71	8.33
	8	64	23.81	21.15	24.29	21.49
800	3	8	10.28	2.50	9.17	6.39
	3	16	13.89	5.50	13.33	6.75
	4	16	15.05	5.65	9.32	3.06
	4	32	20.15	10.33	15.80	11.11
	8	32	18.89	5.50	17.60	13.60
	8	64	22.22	13.14	20.65	19.05
1600	3	8	8.20	2.02	4.85	3.63
	3	16	11.83	3.66	9.95	5.65
	4	16	12.82	3.82	6.97	3.79
	4	32	16.67	6.91	11.29	8.60
	8	32	15.87	7.68	12.58	8.58
	8	64	25.56	7.11	15.33	9.88

Table 6.12. Execution time averages of the solution found by SA,KL,general MFA and hypercube-specific MFA for randomly generated mapping problem instances for various hypercubesize

Problem Size			Average Execution Time(sec)			
TIG		Hypercube	KL	SA	MFA	
N	d_{avg}	$P \times Q$			Gen.	Mesh
400	3	8	0.77	41.27	8.55	0.81
	3	16	1.13	64.57	18.75	2.35
	4	16	1.23	62.49	7.41	1.97
	4	32	2.17	106.25	10.48	6.77
	8	32	1.52	79.87	6.18	3.00
	8	64	2.58	124.63	8.58	4.63
800	3	8	1.26	123.65	7.78	1.49
	3	16	1.91	147.90	15.07	3.99
	4	16	2.15	156.51	7.53	3.20
	4	32	2.95	252.31	15.65	7.19
	8	32	4.37	410.88	15.85	5.45
	8	64	13.62	707.90	44.46	13.26
1600	3	8	2.42	209.69	22.64	2.64
	3	16	0.31	329.72	29.66	7.06
	4	16	3.69	432.32	9.96	5.29
	4	32	5.68	712.89	47.81	17.42
	8	32	8.59	749.02	96.08	14.84
	8	64	16.59	2462.81	241.73	45.38

$100 \times (CL_{max} - CL_{min})/CL_{avg}$. Here, CL_{max} and CL_{min} denotes the maximum and minimum processor loads and CL_{avg} denote the computational loads of processors under perfect load balance conditions. Execution time averages are measured on a DEC Alpha workstation in seconds for randomly generated mapping problem instances.

Chapter 7

CONCLUSION

In this thesis, we try to solve two combinatorial optimization problems, *global routing* problem in design automation of FPGA and *domain mapping* problem in parallel processing, by using *Mean Field Annealing* method.

First of all, Static RAM based Field Programmable gate arrays (FPGA) is modeled as 2-dimensional mesh graph. Then we have proposed an order-independent global routing algorithm, for FPGA based on Mean Field Annealing. The performance of the proposed global routing algorithm is evaluated in comparison with the LocusRoute global router for *ACM/SIGDA* benchmark circuits. Initial experimental results indicate that the proposed MFA heuristic performs better than the LocusRoute.

We proposed an encoding scheme to applied MFA onto global routing problem for FPGA. Our aim is to minimize the energy function of our spin (particles) system. It corresponds to minimize the our objective function, that is finding most uniform distribution routes of the nets (balanced routing). We expected from most uniform distribution of routes that the following detailed routing shows a good performance. (Decrease in total number of segment used, decrease in channel width, and decrease in average delay of nets).

Experimental results show that our expectation was true, the MFA algorithm found more uniform distributed routing that LocusRoute algorithm, therefore the performance of the detailed routing for 100% routing is better in MFA than in LocusRoute for many benchmark circuits.

We have some difficulties in MFA formulation. In this formulation, it is the

first time that Potts spins have different number of states. In Previous MFA formulation for various combinatorial optimization problem, all Potts spins have same number of state, therefore the affect of spin values on the problem remains same but now, as Potts spin vector has different dimension, the affects of spins on problem are different. This may cause some problem therefore we have to find a normalization function that keeps the affect of spins same.

Also if we can find better cooling schedule than we may get better results than we have got. Especially, critical temperature is very important here, if it is initialized to very low temperature, than MFA find a local minimum as a global minimum.

In the second part of this thesis, we have proposed an efficient mapping heuristic for mesh and parallel-connected parallel architecture based on Mean Field Annealing(MFA). We have also developed an efficient implementation scheme for the proposed mapping formulation. The proposed MFA scheme asymptotically reduces the complexity of a single MFA iteration from $\Theta(d_{avg}PQ + (PQ)^2)$ of the general MFA formulation to $\Theta(d_{avg}(P+Q)+PQ)$ for a P by Q mesh. For a square mesh with K processors, this corresponds to an asymptotical complexity reduction from $\Theta(d_{avg}K + K^2)$ to $\Theta(d_{avg}\sqrt{K} + K)$. And for hypercube type architecture complexity of the one MFA iteration is $O(d_{avg} \times \log K + K \times \log K)$ instead of $O(d_{avg} \times K + K^2)$ in the traditional MFA algorithm.

The performance of the proposed mapping heuristic is evaluated in comparison with the well-known heuristics Kernighan-Lin (KL), Simulated Annealing (SA) and general MFA formulation for a number of randomly generated mapping problem instances and Harwell-Boeing sparse matrix test collection. The proposed topology-specific MFA formulation is found to be significantly faster than the general MFA formulation as is expected. The proposed MFA heuristic is slightly slower than the fast KL heuristic. However, it always produces significantly better solutions than the KL heuristic. The quality of the solutions obtained by the proposed MFA heuristic are comparable to those of the powerful SA heuristic. However, the proposed MFA heuristic is orders of magnitudes faster than the SA heuristic. If we can find a good cooling scheduling and initial temperature parameter, then we can get better results. We conclude that for mapping problem, MFA can be located on the algorithms line between the KL and SA.

Bibliography

- [1] *Fundamental of Placement and Routing*. Xilinx Company, SanJose, California, 1990.
- [2] *The Programmable Gate Array Data Book*. Xilinx Company, SanJose, California, 1992.
- [3] S. Brown B. Tseng, J.Rose. Using architectural and cad interactions to improve fpga routing architecture. In *First International Workshop on Field Programmable Gate Arrays*, pages 2-7. ACM, 1992.
- [4] S. H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, 30(3):207-214, 1981.
- [5] T. Bultan. *Parellel mapping and circuit partitioning heuristic on mean field annealing*. PhD thesis.
- [6] T. Bultan and C. Aykanat. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, 16:292-305, 1992.
- [7] F. Ercal C. Aykanat, F. Ozguner and P. Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transactions on Computers*, 37:1554-1567, 1988.
- [8] D. E. Vand den Bout and T. K. Miller. Improving the performance of the hopfield-tank neural network through normalization an annealing. *Biological Cybernetics*, 62:129-139, 1989.
- [9] D. E. Vand den Bout and T. K. Miller. Graph partitioning using annealing neural networks. *IEEE Transaction on Neural Networks*, 1(2):192-203, 1990.

- [10] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.
- [11] R. Francis, J. Rose, and Z. Vranesic. Chortle-crt: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 227–233, 1991.
- [12] J. Lewis I. Duff, R. Grimes. Sparse matrix test problems. *ACM Transaction on Mathematical Software*, 15(1):1–14, march 1989.
- [13] B. Indurkha and H. S Stone. Optimal partitioning of randomly generated distributed programs. *IEEE Transaction on Software Engineering*, 12(3):453–495, 1986.
- [14] S. Kaptanoglu J. Greene, V. Roychowdhury and A. El Gamal. Segmented channel routing. In *International Conference on Computer Aided Design*, pages 567–572. IEEE, 1990.
- [15] F. Ercal P. Sadayappan J. Ramanujam. Task allocation by simulated annealing. In *Proceeding of International Conference on Supercomputing*, pages 475–497, Boston, MA., May 1988.
- [16] A. El Gamal J. Rose and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate-array. *Proceedings of IEEE*, 81(7):1013–1029, july 1993.
- [17] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.
- [18] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
- [19] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley and Sons, Inc., Chichester, West Sussex, England, 1990.
- [20] S. Brown M. Khellah and Z. Vranesic. Minimizing interconnection delays in array-based fpgas. In *Proceedings of Canadian conference on VLSI*, 1994.
- [21] F. Ercal P. Sadayappan and J. Ramanujam. Cluster partitioning approaches to mapping parallel programs onto hypercube. *Parallel Computing*, 13:1–16, 1990.

- [22] C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 3(1):3–22, 1989.
- [23] B. Fallah J. Rose. Timing-driven routing segment assignment in fpgas. In *Proceesings of Canadian Conference on VLSI*, pages 1–7, 1992.
- [24] J. Rose. Parallel global routing for standard cells. *IEEE Transactions on Computer-Aided Design*, 9(10):1085–1095, october 1990.
- [25] Z. Vranesic S. Brown, J. Rose. A detailed router for field-programable gate arrays. In *International Conference on Computer Aided Design*, pages 382–385. IEEE, 1990.
- [26] P. Sadayappan and F. Ercal. Nearest-neighbour mapping of finite element graphs onto processor meshes. *IEEE Transactions on Computers*, 36(12):1408–1424, 1987.
- [27] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, 1993.
- [28] J. Shield. Partitioning concurrent VLSI simulation programs onto a multi-processor by simulated annealing. *IEE Proceedings Part-G*, 134(1):24–28, 1987.
- [29] B.A Hendrickson W. Camp, S. J. Plimpton and R. W. Leland. Massively parallel methods for engineering and science problems. *Communication of ACM*, 37(4):31–41, April 1994.