VOT 72179

PARALLEL PROCESSING MODELS FOR IMAGE PROCESSING PROBLEMS

SHAHARUDDIN SALLEH

BAHROM SANUGI

JABATAN MATEMATIK

FAKULTI SAINS

UNIVERSITI TEKNOLOGI MALAYSIA

2003

DEDICATION AND ACKNOWLEDGEMENT

This research has been conducted mostly at the Department of Mathematics, Faculty of Science, Universiti Teknologi Malaysia. Part of the work was done at the Department of Computer Science, Old Dominion University, USA. The authors would like to thank the University for providing the moral and financial support in the form of the Malaysian Government IRPA Grant no. 72179.

ABSTRACT

This report is the compilation of our research work under IRPA Vot no. 72179. The work consists of four main problems of study. First, we look at the stochastic task scheduling problem using the reconfigurable mesh network as the computing platform. Through our model called the *Dynamic Simulator on Reconfigurable Mesh* (DSRM) which maps a randomly generated number of tasks onto the processors at discrete time, some reasonably good load balancing results were obtained. The second problem is the application of the first model in the edge detection problem using the Laplacian convolution method on the same parallel computing network. In the third problem, we extend the scope to include a strategy for the single-row routing of pins and vias in VLSI design, using our model called the *Enhanced Simulated annealing for Single-row Routing* (ESSR). This model is the parallel implementation of the simulated annealing method, and it generates optimum solutions to the problem. The fourth problem is the extension of the single-row routing problem, where a model has been developed to transform a complete graph into its single-row representation. This last problem has some significant contributions in applications such as scheduling and channel assignment problem in cellular telephone systems.

ABSTRAK

Laporan ini mengandungi kerja-kerja penyelidikan kami di bawah peruntukan IRPA Vot no. 72179. Kajian terbahagi kepada empat masalah utama. Pertamanya, kami mengkaji masalah penjadualan kerja menggunakan rangkaian jaring boleh-konfigurasi. Model kami, dipanggil DSRM (*Dynamic Simulator on Reconfigurable Mesh*), berjaya menghasilkan satu sistem penjadualan yang baik yang memeta kerja-kerja secara rawak dengan pengseimbangan beban. Masalah kedua adalah mengenai aplikasi daripada masalah pertama terhadap masalah pencarian sempadan bagi suatu imej menggunakan teknik konvolusi Laplacian pada rangkaian komputer selari yang sama. Dalam masalah ketiga, kami mengkaji masalah pencarian laluan baris-tunggal dalam pembangunan VLSI, menggunakan suatu kaedah dipanggil ESSR (*Enhanced Simulated annealing for Single-row Routing*). ESSR merupakan model kami yang mengimplementasi kaedah penyelindapan simulasi secara selari, untuk menghasilkan keputusan yang optimum. Masalah keempat merupakan sambungan daripada masalah ketiga, di mana teknik ESSR digunakan untuk menjelma suatu graf lengkap kepada bentuk masalah laluan baris-tunggal.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

SYMBOLS AND NOTATIONS

G	a graph
C_m	a complete graph with m vertices
S_m	single-row representation of C_m
Q	congestion of the nets in S_m
D	number of interstreet crossings (doglegs) in S_m
E	total energy in S_m
L	partial order of nets arranged from top to bottom in S_m
v_j	vertex j in the graph
d_j	degree of vertex j in the graph
m	number of vertices in the graph
t_i	terminal i
b_k	left terminal of net k
e_k	right terminal of net k
n_k	net k, given as $n_k = (b_k, e_k)$
$n_{y,i,m}$	the ith net in level y in S_m
$b_{y,i,m}$	beginning (left) terminal of the ith net in level y in S_m
$e_{y,i,m}$	end (right) terminal of the ith net in level y in S_m
$w_{y,m}$	width of every net in level y in S_m
$r_{y,m}$	number of nets in level y in S_m
z_j	zone j in S_m

CHAPTER 1

RESEARCH FRAMEWORK

1.1 Introduction

Numerous problems in science and engineering today require fast algorithms for implementations and executions on computers. These problems involve massive computations arising from intensive mathematical calculations with double precisions variables and large array sizes. The solutions require high degree of accuracy and constant updates that really take up the maximum capability of the host computers. As a result, single-processor computers based on the von Nerumann architecture seldom satisfy all these requirements. Fast computing platforms with large storage area for processing data in the form of parallel computing networks become the ultimate tools for solving these types of problems.

In our research, we study various number-crunching problems, formulate them as solutions in the form of parallel algorithms and then develop these ideas into their visualization models. The problems of interest in our research include image processing and routing. We develop the parallel algorithms for these problems and their solutions in the form of user-friendly softwares.

1.2 Problem Statement

The problem in this study consists of the development of fast algorithms for highly intractable engineering problems. Three main problems are studied, namely, task scheduling for parallel computing networks, image processing and routing for the VLSI design as some of the applications in task scheduling.

In the first problem, we study the task scheduling problem for the parallel computing network. A task is defined as a unit of job in a computer program. Task scheduling can be stated as the problem of mapping a set of tasks, T_i, onto a set of processing elements, P_k, in a network, with the main objective of completing all the jobs at the most minimum time. In this work, we study the problem of scheduling randomly generated tasks on a reconfigurable mesh network. A mesh network consists of m x n processors arranged in a rectangular array. Each processor in the network has

the processing capability and some storage area for the data. The intermediate processor communicates with its four neighbors through its *east, west, north* and *south* ports.

In the second problem, we study several methods for detecting the edges of an image. An image consists of a rectangular array of pixels, each with a varying degree of intensity represented as colours and gray tone scale. The problem of edge detection can be stated as searching a set of boundary pixels $b(i, j)$ that separate the high and low intensities of the given image, $f(i, j)$. Edge detection is one of the most fundamental components in image analysis, as the method contributes in solving problems such as object recognition, noise reduction, multiresolution analysis, image restoration and image reconstruction. Since an image is normally rectangular in shape, the parallel mesh computing network provides a good platform for its solution. Physically, mesh network provides an ideal tool for solving the image processing problem as each of its processor directly maps the pixels of the given image.

In addition, we also study the routing techniques for the *very large scale integration* (VLSI) design problem in the printed circuit board (PCB). In VLSI, two main problems arise in order to produce a highly compact and integrated design. The first problem is the placement of millions of minute electronic components into the small area of the chip. The second problem deals with the development of routes that connect pairs of these components to allow them to communicate with each other. In this work, we study and develop a model for the second problem based on the single-row routing technique.

1.3 Objectives of Research

The objectives of our research are as follows:

1. To promote fundamental research that integrates mathematics with its applications, especially in areas of engineering and information technology.
2. To develop and promote parallel algorithms and solutions on highly interactive combinatorial problems, and its simulation and visualization models.
3. To promote learning groups on various problems of this nature in the community.
4. To contribute the ideas to the interested parties in industries for further collaboration.

1.4 Scope of the Study

Our study is confined to the development of simulation models for task scheduling, edge detection in image processing and routing problems based on the mesh network. The work extends to the development of algorithms and user-friendly computer softwares based on the personal computer Microft Windows environment.

1.5 Report Outline

The report is organized into seven chapters. Chapter one is the research framework where the problems, objectives and scope of the work are described.

In Chapter two, we describe an overview of the overall parallel computing system, some common topologies and ideas for the processor parallelization. One particular interest of the parallel computing system is the reconfigurable mesh network. We

discuss the architecture of the reconfigurable mesh which has ports that can be configured dynamically according to the requirements of the program.

In Chapter three, we discuss the task scheduling problem on the reconfigurable mesh network. Task scheduling is a combinatorial optimization problem that is known to have large interacting degrees of freedom and is generally classified as NP-complete. Task scheduling is defined as the scheduling of tasks or modules of a program onto a set of autonomous processing elements (PEs) in a parallel network, so as to meet some performance objectives.

In Chapter four, we present the edge detection method which is an application of the task scheduling problem. Edge detection is a technique of getting a boundary line which holds the key to other image processing requirements, such as object recognition, image segmentation, and image analysis. We concentrate on the development of the second-order Laplacian convolution technique on the mesh network for this problem.

Chapter five discusses another application of task scheduling, namely, the single-row routing problem. In the single-row routing problem, we are given a set of n evenly spaced terminals (pins or vias) arranged horizontally from left to right in a single row called the node axis. The problem is to construct nets in the list from the given intervals according to the design requirements. In this chapter, a model called the Enhanced Simulated annealing for Single-row Routing (ESSR) is proposed to represent the solution to the problem.

In Chapter six, we formulate the concept of transforming a complete graph into its single-row representation. This idea is a significant contribution in the sense that it generalizes the single-row routing as an effective application from other applications. Through this technique, any problem that can be represented by a graph is reducible to its the single-row routing representation.

Finally, Chapter six is the conclusion and suggestions for further research.

CHAPTER 2

RECONFIGURABLE MESH COMPUTING NETWORKS

2.1 INTRODUCTION

Observation, theory, and experimentation are basic action for classical science. All these will lead to a hypothesis. From that, scientists will develop a theory to explain the phenomenon and design a physical experiment to test the theory. Usually the results of the experiment require the scientists either to refine the theory or completely reject it. And the process will repeat again and again. All this experiments may be too expensive or time consuming. Some may be unethical or impossible to perform.

Contemporary science, then, is characterized by observation, theory, experimentation and numerical simulation. Numerical simulation is an increasingly important tool for scientists. Many important problems are just too complex that solving them via numerical simulation requires extraordinarily computers. High speed computers allow scientist to test their hypotheses in another way by developing a

numerical simulation of a phenomenon. Instead of doing physical experiments, they can save time through effective simulations.

The followings are some of the several categories of complex problems (Levin 1989) that require massive numerical computations:

1. Quantum chemistry, statistical mechanics, and relativistic physics.
2. Cosmology and astrophysics.
3. Computational fluid dynamics and turbulence.
4. Biology, pharmacology, genome sequencing, genetic engineering, protein folding, enzyme activity and cell modeling.
5. Global weather and environmental modeling.

These entire problems can be solved by the fastest computer in the world which is built of numerous microprocessors. This computer is also known as parallel computer. In order to keep up to this high speed computing, studying parallel algorithms is a necessity today.

This chapter is divided into five sections. Section 2.1 is the introduction, while in Section 2.2, we discuss the importance of parallel computers. A good analogy is presented to make the problem easier to understand. Section 2.3 in this chapter reviews the paradigms of parallel processing. From the Flynn's taxonomy , the architecture of a parallel can be classified as SISD, SIMD, MISD, and MIMD. The next section discusses the memory models of computation which is divided into serial and parallel. In a serial computational model, the model is called *random access machine* (RAM), while in parallel, it is called *parallel random access machine* (PRAM). Section 2.5 presents the topology of the network, which is the way processors are organized. The last section is about the Reconfigurable Mesh network. We discuss the architecture of the reconfigurable mesh, the differences between a reconfigurable mesh network and the ordinary mesh network, and lastly an example on the application of reconfigurable mesh in sorting numbers.

2.2 WHY DO WE NEED PARALLEL COMPUTERS?

The solution to a typical numerical problem in engineering today requires the use of several large size multi-dimensional arrays, multi-level loops and the thousands of lines of code, in a single program. As a result, the program needs to be written in a very systematic manner, with proper software engineering techniques and implementations. The burden of a single computer system is greatly reducing by distributing the load to the processors in the system. As a result, the individual processors are not too overloaded and the same amount of work can be completed in a much faster time with a network of cooperating processors.

A computer, as described in Zomaya (1996), is a digital electronic device with either a sequential or parallel design. A *sequential computer* is a random access memory model (RAM) that contains one processing element (processor) and an attached main-memory unit in an architecture known as the von Neumann design. This digital machine reads and executes instructions and data sequentially using only one processor. In contrast, a *parallel computer* consists of a set of at least two computing elements, all of which are connected in a network so that each one of them will be able to communicate, and share resources and energy with others in performing a job. The parallel counterpart to the RAM model, called the parallel random access memory (PRAM), has a set of synchronous processors connected to a shared memory.

Much of the original contributions to the parallel processing ideas evolves from the Kolmogorov's Theorem, presented as follows:

Kolmogorov's Theorem (Kolmogorov, 1957): Any continuous function $f(x_1,x_2,...,x_n)$ of n variables $x_1,x_2,...,x_n$ on on the interval I_n can be represented in the form

$$f(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{2n+1} h_j \left(\sum_{i=1}^{n} g_{ij}(x_i) \right) \qquad (2.1)$$

where h_j and g_{ij}'s are continuous functions of one variable. Furthermore, the g_{ij}'s are fixed, monotonic increasing functions that are not dependent on $f(x_1,x_2,...,x_n)$. This theorem provides a very useful development of parallel algorithms that relate a problem with its solution in implicit or explicit manner.

A good analogy we can use to describe how the serial and parallel computers work is in the construction of a house. If there is only one worker who will do all the entire job (bricklaying, plumbing, and installing wiring, etc) by himself, he is going to take a very long time to finish a house. All the tasks will be done one by one in a sequence and this is called the sequential approach which is very slow way. However, by splitting the tasks to several workers, the construction can be completed much faster. The workers can be assigned different and independent tasks simultaneously, and this contributes to faster completion.

If we compare it with modern computers today, we can see that a computer with a single processor are most likely the house constuction with only a worker. This single processor which does the computational work such as the addition, multiplication, and the comparison of two numbers. Programmers divide the computational work into a few sequence steps (a program) and the sequence will be executed step by step. This is surely a very slow way to execute a task. Figure 2.1 shows a conventional computer based on a single processor that illustrates this classical idea.

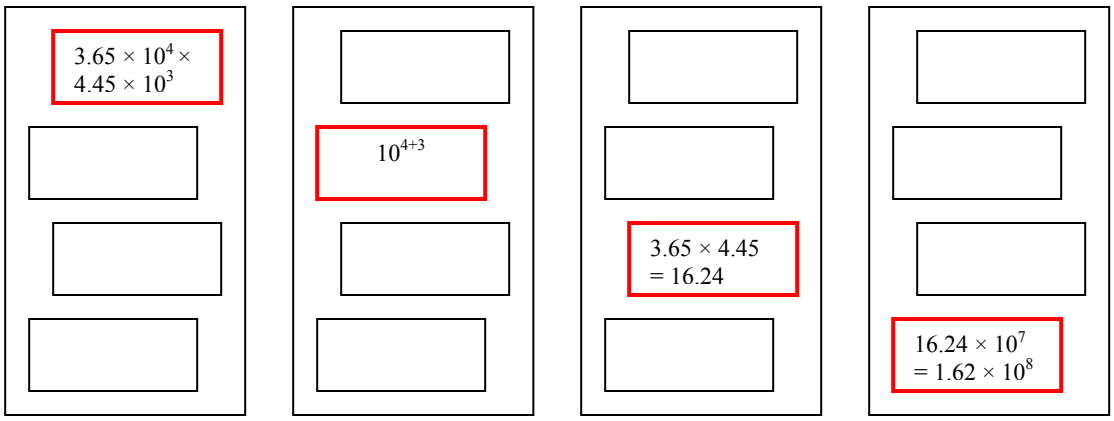


Figure 2.1: a conventional processor

This reason has lead many computer designers to develop the solution for this problem. The slowness of the computer in executing a task are caused by the access to memory. When the data is fetched from the memory, all the processor's functional unit that perform the computation must remain idle. After executing, the result must be sent to the memory and again, this will involve some extra overhead. Another problem arises when the processor needs to fetch more than one operand at the same time. While the first operand is fetched, the second operand must wait until the first has completed its job. A solution to this problem lies in a co-operative system called interleaved memory. Interleaved memory consists of a small number of separately accessible memory units. In this system, several units of memory can be accessed at the same time through separate channels. Data too can be fetched without having to wait for channels to clear first.

Another reason that causes the slowness in a computer is the tedious process of computations. Imagine a very large number to be multiply with another large number, of course, it will take a few small steps before the computation can be done. In a conventional computer, this step is done in a way which cause some processor idle while waiting for a task to be executed. A pipelined processor, as shown in Figure 2.2,

can be used to solve this problem. It is effective for applications that require many repetitions of the same operation.

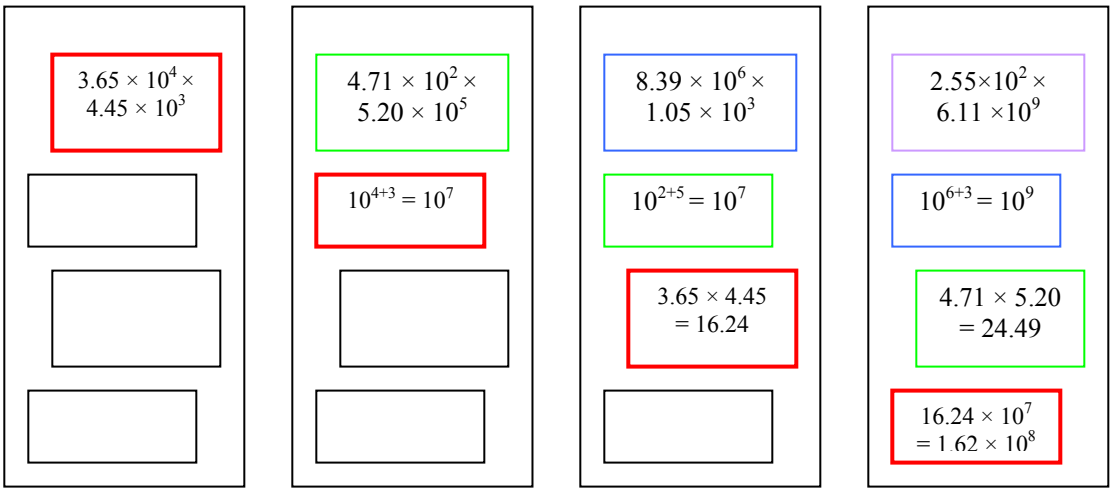


Figure 2.2: A pipelined processor

The same house construction anology can also apply in parallel computer. It seems like parallel when there are many workers doing different parts of the job. What makes the system is good is because each individual has several function unit. All the workers are differentiate by their speciality either in doing bricklaying, plumbing or wiring. This system is also said to have a coarse grain size because the tasks assigned to each worker are in certain amount. These people are similar with processors in the parallel computer.

The laborers are communicated with each other. For example, bricklayers working next to each other must stay in close communication to make sure the build a uniform wall between section. This is called the nearest-network topology. However, such a system can lead to overhead because while sending the message to each other, the workers may talk a lot and do less job. That is why there must be another good topology that can overcome this bottleneck.

2.3 PARALLEL PROCESSING PARADIGMS

Computers operate simply by executing a set of instructions on a given data. A stream of instructions inform the computer of what to do at each step. From the concepts of instruction stream and data stream, Flynn (Flynn, 1972) classified the architecture of a computer into four. **Instruction stream** is a sequence of instructions performed by a computer; a **data stream** is a sequence of data manipulated by an instruction stream. The four classes of computers are:

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data stream (SIMD)
- Multiple instruction stream, single data stream (MISD)
- Multiple instruction stream, multiple data stream (MIMD)

2.3.1 Single instruction stream, single data stream (SISD)

Most serial computers belong to the SISD class that have been designed based on the von Neumann architecture. In such computers, the instructions are executed sequentially which means the computer executes one operation at a time. The algorithm used in this class is known as a sequential algorithm. Although a SISD computer may have multiple functional units, there are still under the direction of a single control unit. Figure 2.3 illustrates a SISD computer.

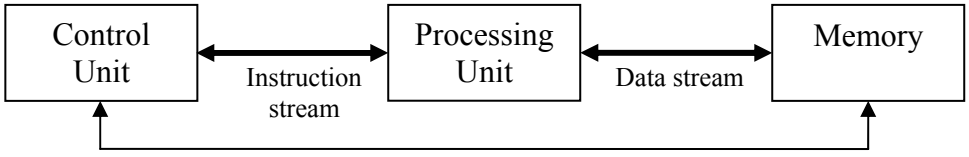


Figure 2.3: SISD

2.3.2 Single instruction stream, multiple data stream (SIMD)

SIMD machines consist of *N* processors, *N* memory, an interconnection network and a control unit. All the processor elements in the machine are supervised by the same control unit. These processors will be executing the same instruction at the same time but on different data. In terms of memory organization, these computers are classified as *shared memory* or *local memory*. To get an optimal performance, SIMD machines need a good algorithm to manipulate many data by sending instruction to all processors. Processor arrays fall into this category.

2.3.3 Multiple instruction stream, single data stream (MISD)

Among all four, MISD is the least popular model for building commercial parallel machine. Each processor in MISD machine has its own control unit and shares a common memory unit where data reside. Parallelism is realized by enabling each processor to perform a different operation on the same data at the same time. Systolic arrays are known to belong to this class of architectures. *Systolic* means a rhythmic contraction. *A systolic array* is a parallel computer that rhythmically 'pumps' data from processor to processor. The might be some changes in the data everytime it goes through the processors because each processor may modify the data before passing it to the next processor. Figure 2.4 shows a typical network based on the SIMD model.

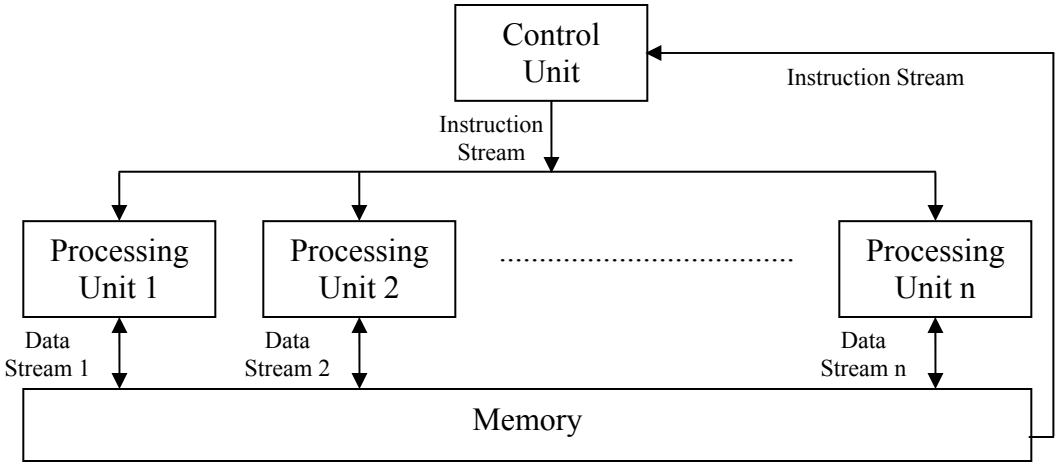


Figure 2.4: SIMD Model

2.3.4 Multiple instruction stream, multiple data stream (MIMD)

MIMD machines are the most general and powerful system that implements the paradigm of parallel computing. In MIMD, there are *N* processors, *N* streams of instructions and *N* streams of data. As shown in Figure 2.5, each processor in MIMD has its own control unit.

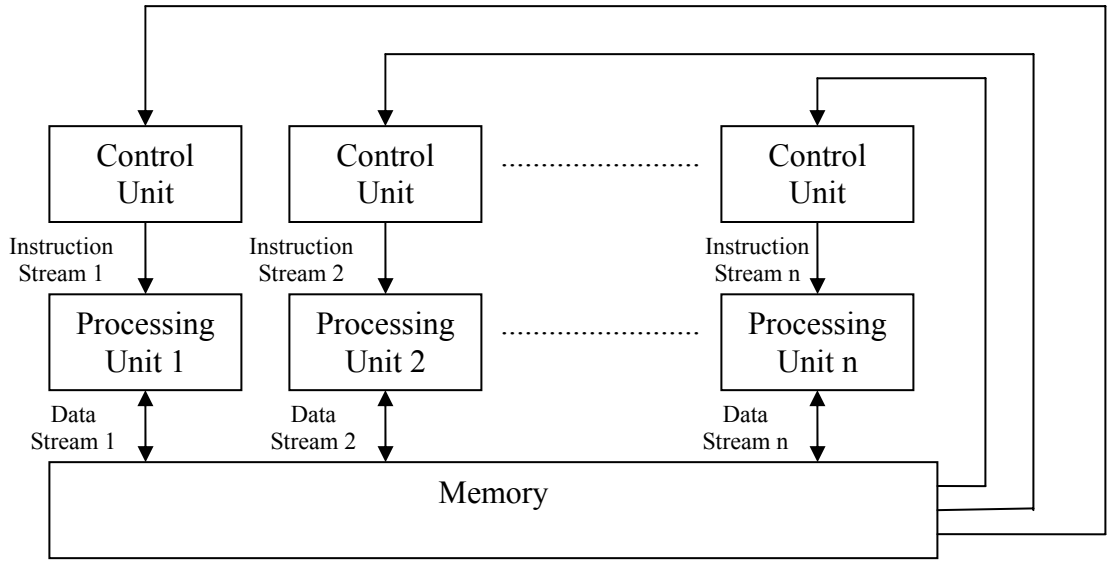


Figure 2.5: MIMD Model

2.4 MEMORY COMPUTATIONAL MODELS

There are two models of computation. First is the serial model of computation while the other one is the parallel model of computation. The *random access memory* (RAM), as shown in Figure 2.6**,** is the sequential model of computation. The model consists of a memory, a read-only input tape, a write-only output tape, and a program.

.

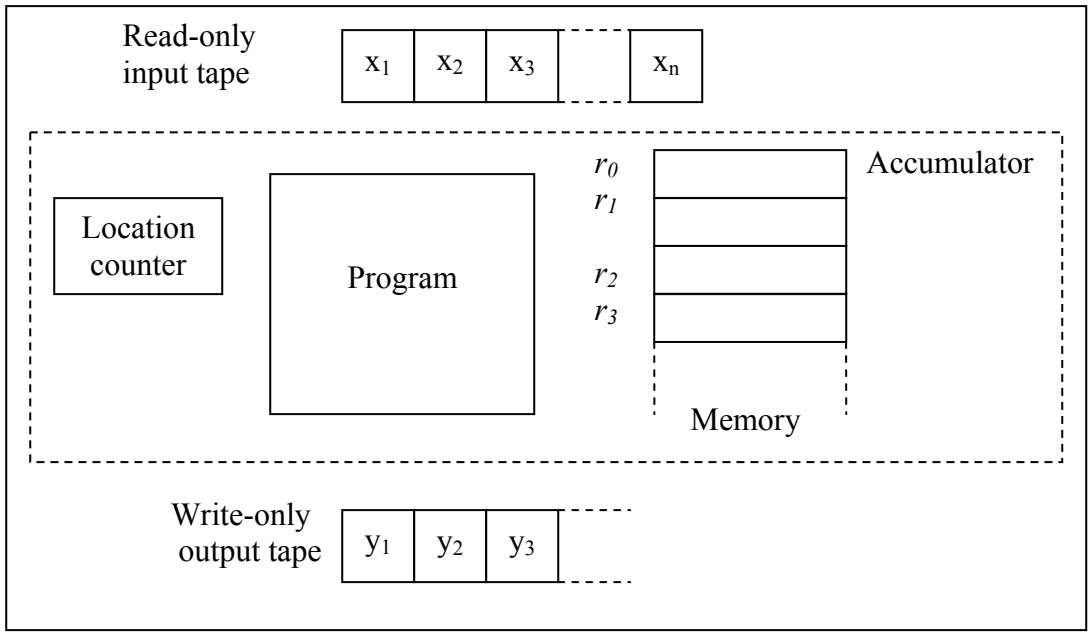


Figure 2.6: The RAM model of sequential computation

Parallel processing actually is information processing that emphasizes the concurrent manipulation of data elements belonging to one or more processors solving a single problem. While a parallel computer is a multiple processor computer capable of parallel processing. A theoretical model for parallel computation is the *parallel random access machine* (PRAM).

The PRAM model, as shown in Figure 2.7, allows parallel-algorithm designers to treat processsing power as an unlimited resource, much as programmers of computers with virtual memory are allowed to treat memory as an unlimited resource. It is unrealistically simple which means it ignores the complexity of interprocessor communication. By doing that, it can focus on the parallelism inherent in a particular computation. A PRAM consists of a control unit, global memory, and an unbounded set of processors, each with it's own private memory. A PRAM computation begins with the input stored in global memory and a single active processing element. During each step of the computation an active, enabled processor may read a calue from a single private or global memory location, perform a single RAM operation and write into one local or may activate another processor.

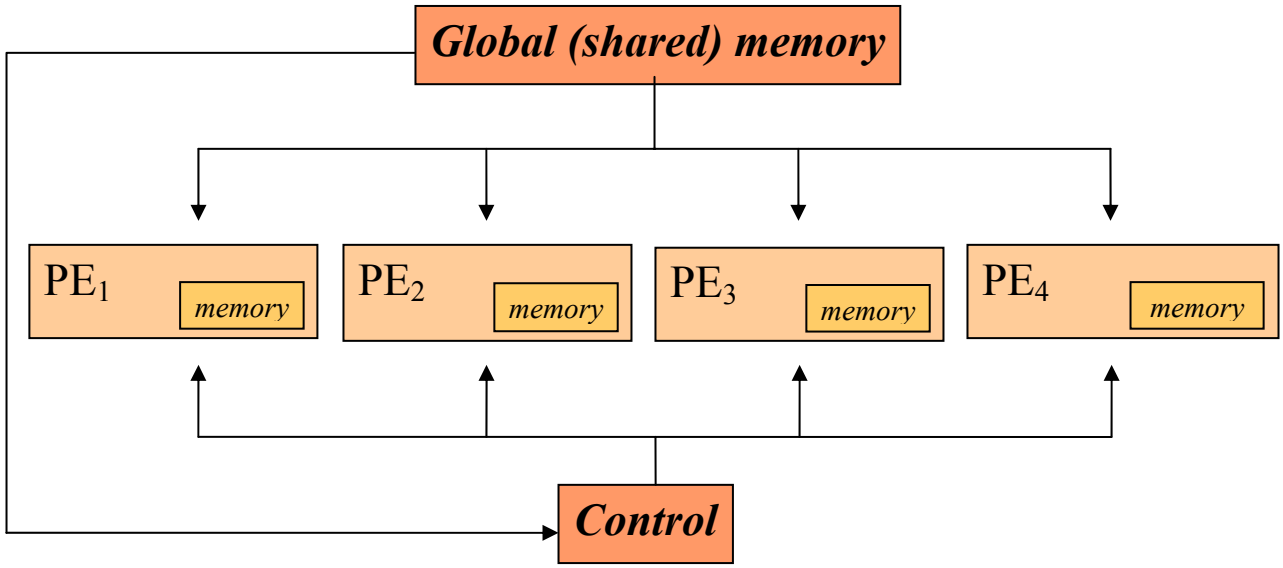


Figure 2.7: The PRAM model of parallel computation

2.5 PROCESSOR ORGANIZATIONS / TOPOLOGY

The topology of a network describes how processors are distributed and organized in it. In terms of graph, the processors are represented as nodes and the edges linking any pair of nodes in the graph are the communication links between the two processors (El-Rewini et al., 1992). Some common types of processor organizations include the mesh, binary trees, hypertree, butterfly, pyramid, hypercubes, shuffle-exchange and the De Bruijn model.

These processor organizations are evaluated according to criteria that help determine their practicality and versatility. The criteria include:

1. **Diameter**
   The diameter of a network is the largest distance between two nodes.

2. **Bisection width**
   The bisection width of a network is the minimum number of edges that must be removed in order to divide the network into two halves.

3. **Number of edges per nodes**
   The number of edges per nodes should be maintained as a constant independent of the network size. This is because it would be easier for the processor organizations to scale the system with large number of nodes.

4. **Maximum edge length**
   It is best if the nodes and edges of the network can be laid out in three dimensional space. By doing this, the maximum edge length can be a constant independent of the network size.

In this chapter, we focus on the reconfigurable mesh (RMesh) network as a platform for solving the task scheduling, image processing and routing problems. Mesh-based architectures have attracted strong interest because of the following reasons:

- The wiring cost is cheap as the complexity is lower compared to other models, such as the hypercube.
- It has a close match for direct mapping on many application problems, such as in task scheduling and image processing.

A regular mesh of size $N \times N$ has a communication diameter equals to $2\,(N-1)$. The time needed by this network to solve problem like comparing or combining data is $O\,(N)$. To improve the time complexity, that is to get the most minimun computation time, researchers have studied a new architecture based on a 2 or 3 dimensional mesh which provide additional communication links between the processors of the mesh.

2.6 RECONFIGURABLE MESH NETWORK (RMESH)

Reconfigurable mesh is a theoretical parallel computing model which is being used to develop parallel algorithms independent of the hardware factors (Miller and Prasanna-Kumar, 1993). Several fast algorithms for the reconfigurable mesh networks have been developed, among others, as in Stout (1992) for the padded sort problem, and Olariu *et al*. (1993) for the component labelling and convex hull construction problem . These applications contribute in the design of high-performance central processing units (CPU) and other very large-scale integration VLSI circuits. A suitable realization for this model is the message-passing transputer-based system where each transputer represents a processing element with memory module each, and has communication links with other transputer. The architectures allow the network topology to change dynamically as required by the algorithm. As the result, the interprocesser communication will be more flexible and efficient.

2.6.1 Reconfigurable Mesh Architecture

In a reconfigurable mesh network, the processors are arranged into *n*-dimensional arrays of processors (Olariu *et al*., 1993). Figure 2.8 shows a 2-dimensional RMesh. Torus, as shown in Figure 2.9 occurs when wraparound connection are present. Wraparound connection means the connection of the processors at the edge with processors at the another edge of the same row or column. For example, processors on the first row are connected through their north ports to the south ports of processors in the last row.

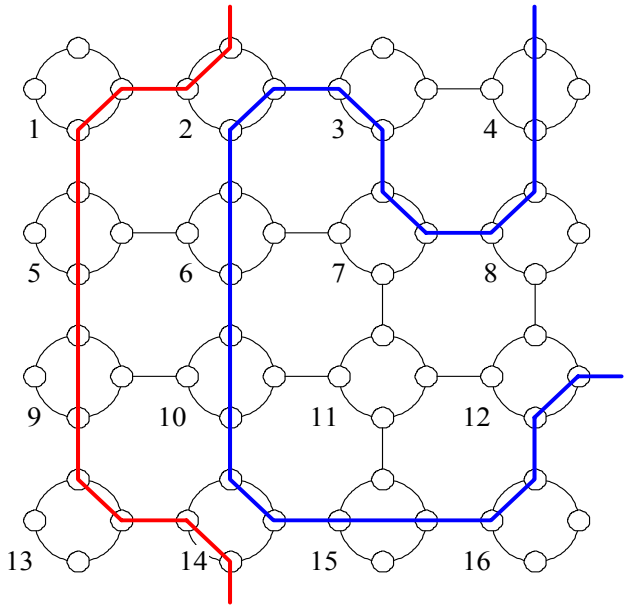


Figure 2.8: 2-dimensional RMesh with 16 processing elements

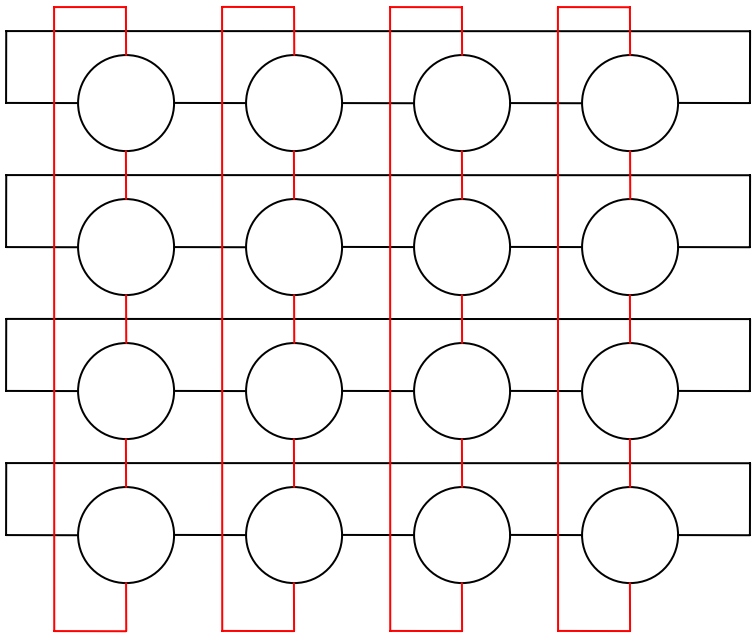


Figure 2.9: Torus

The computational model in Figure 2.8 shows a 4×4 network of 16 processing elements, PE[k], for k = 1, 2, …., 16. For the *n*-dimensional mesh, each processing element in the network has $2n$ external ports. As this model is a two-dimensional network, each processing element has 4 external ports, namely, 'North', 'South', 'East' and 'West'.

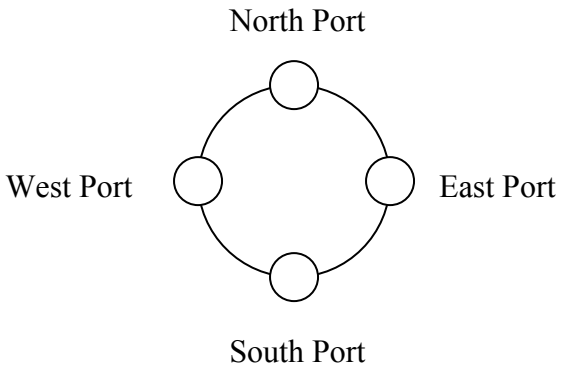


Figure 2.10: Four external ports for a node in a 2-dimensional RMesh network

Figure 2.10 shows a processor in the Reconfigurable Mesh network. Communication between the processing elements in the reconfigurable mesh can be configured dynamically in one or more buses. A **bus** is a doubly-linked list of processing elements, with every processing element on the bus being aware of its immediate neighbors. A bus begins in a processing element, pass through a series of other processing elements and ends in another processing element. A bus that passes through all the processing elements in the network is called the *global bus*, otherwise it is called a *local bus.*

Figure 2.8 shows two local buses $B(1) = \{2,1,5,9,13,14\}$ and $B(2) = \{12,16,15,14,10,6,2,3,7,8,4\}$, where the numbers in the lists represent the processing element numbers arranged in order from the first (starting) processing element to the last (end). As an example from the figure, communication between $PE[9]$ and $PE[13]$ on the bus $B(1)$ is made possible through the link $\{PE[9].s, PE[13].n\}$.

The processing elements in a bus cooperate to solve a given problem by sending and receiving messages and data according to their controlling algorithm. A *positive direction* in a bus is defined as the direction from the first processing element to the last processing element, while the *negative direction* is the opposite. Note that the contents in the list of each bus at any given time t can change dynamically according to the current computational requirements.

2.6.2 Data Transmission in Mesh Networks

We illustrate the idea of data transmission through the diagrams in Figures 2.11a and 2.11b. In this mesh network, data transmission from PE[1] to PE[16] requires 6 hops. The path can be written as follows:

PE [1] → PE [5] → PE [9] → PE [13] → PE [14] → PE [15] → PE [16]

It can be seen that the path needs $2\,(N-1)$ or 6 steps for data transmission.

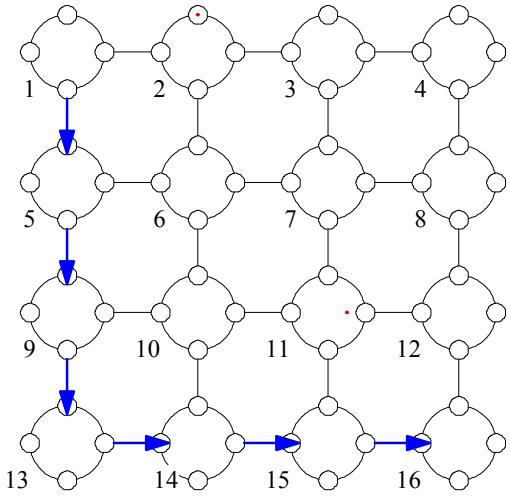


Figure 2.11a: 2D mesh with fixed connection

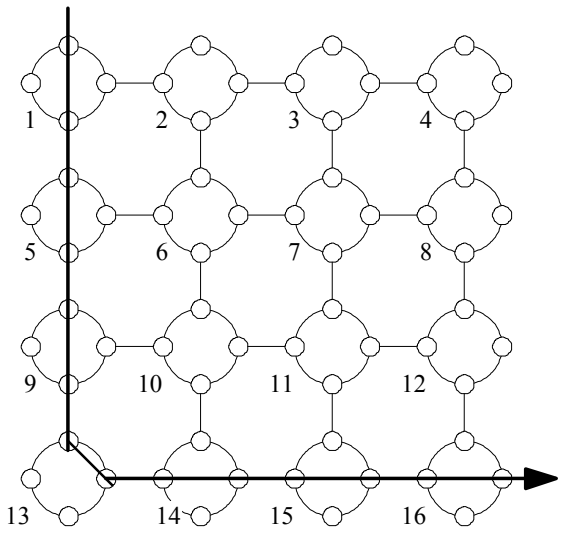


Figure 2.11b : 2D reconfigurable mesh

However, with the reconfigurable mesh, data transmission between the two processors reduces to a constant time. Only two steps are needed by a RMesh network for the same communication between PE [1] to PE [16]. First, it needs to set the switches and recognize which port can be connected with the next processor. Second step is transfering the data by a local bus. In this case, the local bus can be writtan as $B\,(1) = \{1, 5, 9, 13, 14, 15, 16\}$

Reconfigurable mesh network is created in order to provide the flexibility to change the interconnection pattern. So, it is more dynamic and easy to use. While for mesh network, it is a static network. What makes RMesh network dynamic is because of the switches it got in every Processing Elements or nodes. This switches are also known as external ports. For 2-dimensional network, there are four ports that is North, South, East and West.

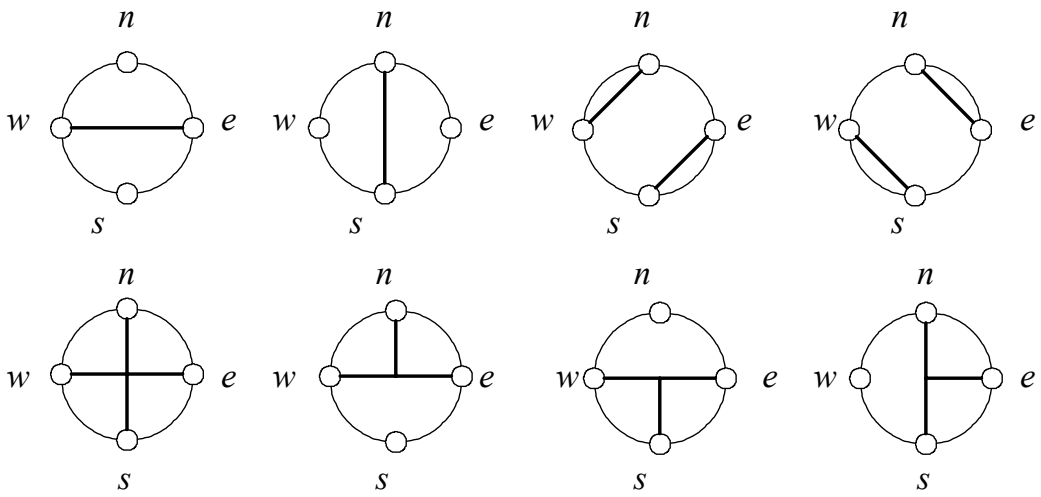


Figure 2.13: A few patterns of the switch connection.

2.7 Sorting Algorithm Example on RMesh

In this section, we illustrate the operation of a reconfigurable mesh network through an example in solving the sorting problem. Given a set of numbers, 6, 9, 1, 5, we need to sort these numbers in ascending order using the Rmesh model. The solution is outlined as follows:

Step 1

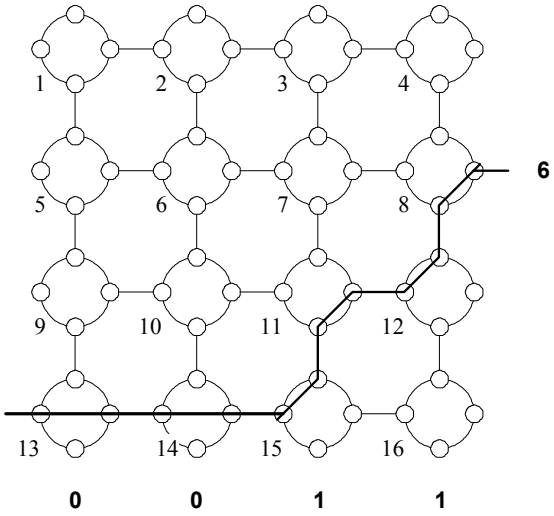
Make a table with these numbers. Compare the numbers in the first column with numbers in the first row. If value of the numbers in the row are same or bigger than the value of the numbers in the column, tick 0 in the box. But if not, tick 1.

Table 2.1: Numbers to sort

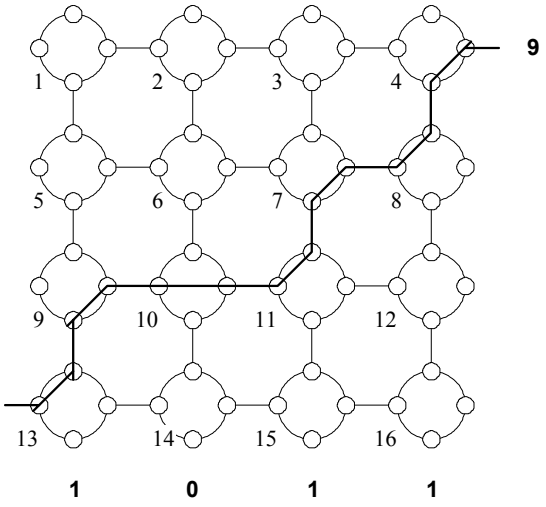
	6	9	1	5
6	0	0	1	1
9	1	0	1	1
1	0	0	0	0
5	0	0	1	0

Step 2

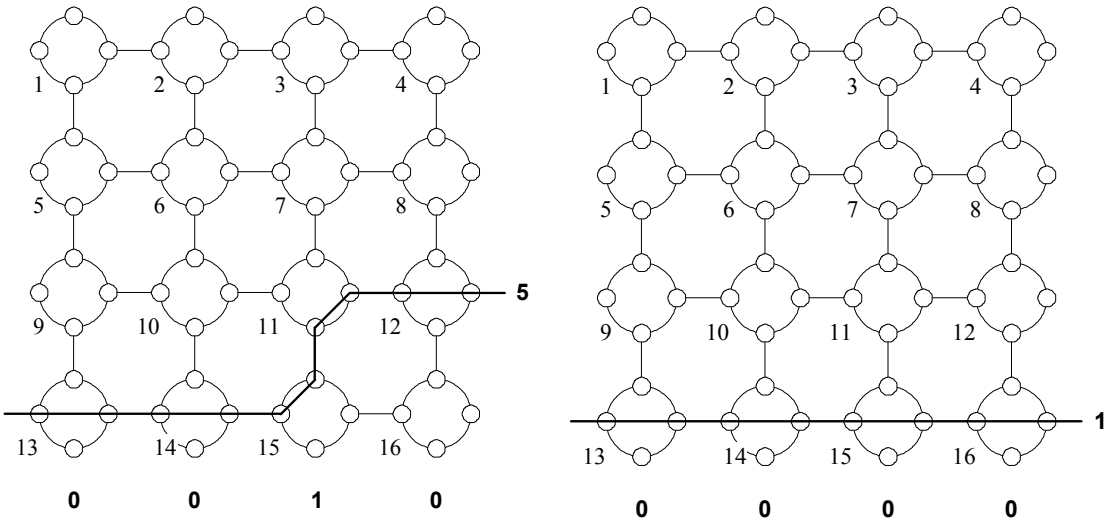
Now that we have the binary number, we can draw the Reconfigurable Mesh network. For 0, we draw a horizontal line through the nodes. While for 1, we draw a vertical line. Figure 2.14 illustrates the steps in solving this problem on Rmesh.



Number 6 at the second rank



Number 9 at the first rank



Number 5 at the third rank

Number 1 at the last rank

Figure 2.14: Numbers in descending order from above

2.8 Summary

This chapter is an overview of the parallel computing systems. One particular network of interest is the reconfigurable mesh system. Rmesh is made up of a rectangular array of processors where each processor has four ports configured dynamically according to the program requirements. We discuss the practicality of Rmesh in solving several number-crunching applications, such as in image processing and task scheduling.

CHAPTER 3

DYNAMIC MULTIPROCESSOR SCHEDULING ON RMESH

3.1 Introduction

Task scheduling is a combinatorial optimization problem that is known to have large interacting degrees of freedom and is generally classified as NP-complete (El-Rewini *et al*., 1994). Most solutions to the problem have been proposed in the form of heuristics. These include approaches using list scheduling, queueing theory, graph theoretic and enumerated search. Task scheduling is defined as the scheduling of tasks or modules of a program onto a set of autonomous processing elements (PEs) in a parallel network, so as to meet some performance objectives. The main objective in task scheduling is to obtain a scheduling model that minimizes the overall execution time of the processing elements. Another common objective is to distribute the tasks evenly among the processing elements, an objective known as *load balancing*. Task scheduling applications can be found in many areas, for example, in real-time control of robot manipulators (Hwang *et al*., 1989), flexible manufacturing systems (Ramamritham and Stankovic, 1989), and traffic control (Ramamritham and Stankovic, 1989).

In terms of implementation, task scheduling can be classified as either static or dynamic. In static scheduling, all information regarding the states of the tasks and the processing elements are known beforehand prior to scheduling. In contrast, all this information is not available in dynamic scheduling and it is obtained *on the fly*, that is, as scheduling is in progress. Hence, dynamic scheduling involves extra overhead to the processing elements where a portion of the work is to determine the current states of both the tasks and the processing elements.

In this chapter, we consider the task scheduling problem on the reconfigurable mesh architecture. A *reconfigurable mesh* is a bus-based network of N identical PE[k], for $k = 1,2,...,N$, positioned on a rectangular array, each of which has the capability to change its configuration dynamically according to the current processing requirements. Figure 3.1 shows a $4 \mathbf{x} 5$ reconfigurable mesh of 20 processing elements. Due to its dynamic structure, the reconfigurable mesh computing model has attracted researchers on problems that require fast executions. These include numerically-intensive applications in computational geometry (Olariu *et. al*, 1994), computer vision and image processing (Olariu *et. al*, 1995) and algorithm designs (Nakano and Olariu, 1998).

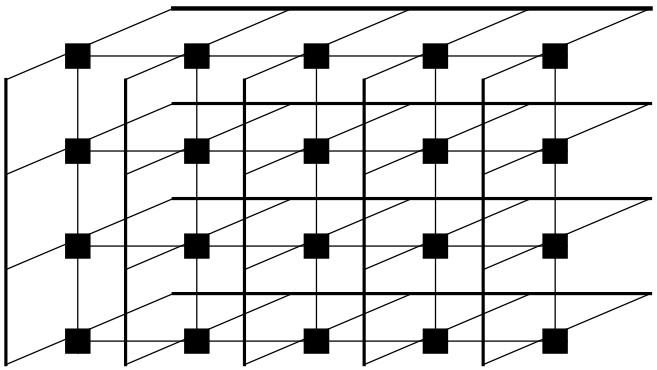


Figure 3.1: A reconfigurable mesh of size $4 \mathbf{x} 5$

This chapter is organized into five sections. Section 3.1 is the introduction. Section 3.2 is an overview of the dynamic scheduling problem, while Section 3.3 describes our model which is based on the reconfigurable mesh computing model. The simulation results of our model are described in Section 3.4. Finally, Section 3.5 is the summary and conclusion.

3.2 Dynamic Task Scheduling Problem

Dynamic scheduling is often associated with real-time scheduling that involves periodic tasks and tasks with critical deadlines. This is a type of task scheduling caused by the *nondeterminism* in the states of the tasks and the PEs prior to their execution. Nondeterminism in a program originates from factors such as uncertainties in the number of cycles (such as loops), the and/or branches, and the variable task and arc sizes. The scheduler has very little *a priori* knowledge about these task characteristics and the system state estimation is obtained on the fly as the execution is in progress. This is an important step before a decision is made on how the tasks are to be distributed.

The main objective in dynamic scheduling is usually to meet the timing constraints, and performing load balancing, or a fair distribution of tasks on the PEs. Load balancing improves the system performance by reducing the mean response time of the tasks. In Lin and Raghavendran (1991), load balancing objective is classified into three main components. First, is the *information rule* which describes the collection and storing processes of the information used in making the decisions. Second, is the *transfer rule* which determines when to initiate an attempt to transfer a task and whether or not to transfer the task. Third, is the *location rule* which chooses the PEs to and from which tasks will be transferred. It has been shown by several researchers that with the right policy to govern these rules, a good load balancing may be achieved.

Furthermore, load balancing algorithms can be classified as *source-initiative* and *server-initiative* (Lin and Raghavendran, 1991). In the source-initiative algorithms, the hosts where the

tasks arrive would take the initiative to transfer the tasks. In the server-initiative algorithms, the receiving hosts would find and locate the tasks for them. For implementing these ideas, a good load-balancing algorithm must have three components, namely, the information, transfer and placement policies. The information policy specifies the amount of load and task information made available to the decision makers. The transfer policy determines the eligibility of a task for load balancing based on the loads of the host. The placement policy decides which eligible tasks should be transferred to some selected hosts.

Tasks that arrive for scheduling are not immediately served by the PEs. Instead they will have to wait in one or more queues, depending on the scheduling technique adopted. In the *first-in-first-out* (FIFO) technique, one PE runs a scheduler that dispatches tasks based on the principle that tasks are executed according to their arriving time, in the order that earlier arriving tasks are executed first. Each dispatch PE maintains its own waiting queue of tasks and makes request for these tasks to be executed to the scheduler. The requests are placed on the schedule queue maintained by the scheduler. This technique aims at balancing the load among the PEs and it does not consider the communication costs between the tasks. In Chow and Kohler (1979), a queueing model has been proposed where an arriving task is routed by a task dispatcher to one of the PEs. An approximate numerical method is introduced for analyzing two-PE heterogeneous models based on an adaptive policy. This method reduces the task turnaround time by balancing the total load among the PEs. A central task dispatcher based on the single-queue multiserver queueing system is used to make decisions on load balancing. The approach is efficient enough to reduce the overhead in trying to redistribute the load based on the global state information.

Several *balance-constrained* heuristics, such as in Saletore (1990), consider communication issues in balancing the load on all PEs. The approach adds balance constraint to the FIFO technique by periodically shifting waiting tasks from one waiting queue to another. This technique performs local optimization by applying the steepest-descent algorithm to find the minimum execution time. The proposed *cost-constraint* heuristic further improves the load balancing performance by checking the uneven communication cost and quantify them as the time needed to perform communication.

Our performance index for load balancing is the *mean response time* of the processing elements. The response time is defined as the time taken by a processing element to response to the tasks it executes. In general, load balancing is said to be achieved when the mean response time of the tasks is minimized. A good load balancing algorithm tends to reduce the mean and standard deviation of the task response times of every processing elements in the network.

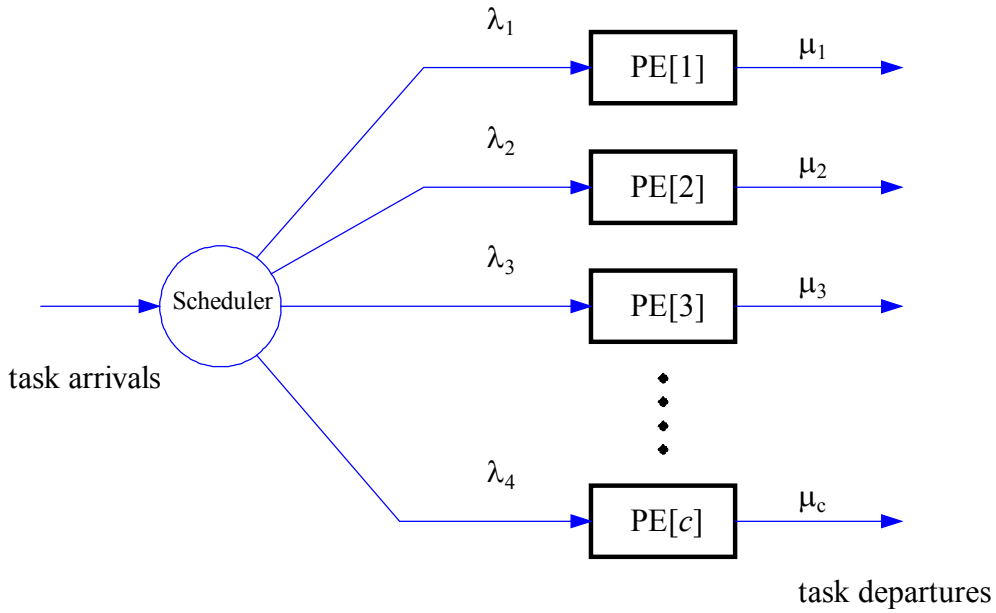


Fig.3.2: The *m/m/c* queueing model

In our work, task scheduling is modeled as the $m/m/c$ Markovian queueing system. An algorithm is proposed to distribute the tasks based on the probability of a processing element receiving a task as the function of the mean response time at each interval of time and the overall mean turnaround time. Tasks arrive at different times and they form a FIFO queue. The arrival rate is assumed to follow the Poisson distribution with a mean arrival rate of λ. The service rate at processing element k is assumed to follow the exponential distribution with mean μ_k. Our idea is illustrated through a simulation model called DSRM which is explained in Section 3.4.

In general, the mean response time R for tasks arriving at a processing element is given from the Little's law defined in (Kobayashi, 1978), as follows:

$$R = \frac{N}{\lambda} \tag{3.1}$$

where N is the mean number of tasks at that processing element. In a system of n processing elements, the mean response time is given as follows (Kobayashi, 1978):

$$R_k = \frac{1}{\mu_k - \lambda_k} \tag{3.2}$$

where λ_k is the mean arrival rate and μ_k is the mean service rate at the processing element k. It follows that the mean response time for the whole system is given as follows (Kobayashi, 1978):

$$R = \frac{1}{n^*} \sum_{k=1}^{n} R_k \tag{3.3}$$

where $n^* = \sum_{k=1, \lambda_k \neq 0}^{n} 1$.

3.3 Reconfigurable Mesh Computing Model

Our computing platform consists of a network of 16 processing elements arranged in a reconfigurable mesh. A suitable realization for this model is the message-passing transputer-based system where each node in the system is a processor which includes a memory module. In addition, each processor in the system has communication links with other processors to enable message and data passing.

3.3.1 Computational Model

The computational model is a 4 **x** 4 network of 16 processing elements, $PE[k]$, for $k = 1,2,...,16$, as shown in Figure 3.3. Each processing element in the network has four ports, denoted as $PE[k].n$, $PE[k].s$, $PE[k].e$ and $PE[k].w$, which represent the north, south, east and west communicating links respectively. These ports can be dynamically connected in pairs to suit some computational needs.

Communication between the processing elements in the reconfigurable mesh can be configured dynamically in one or more buses. A bus is a doubly-linked list of processing elements, with every processing element on the bus being aware of its immediate neighbours. A bus begins in a processing element, pass through a series of other processing elements and ends in another processing element. A bus that passes through all the processing elements in the network is called the *global bus*, otherwise it is called a *local bus.* Figure 3.3 shows two local buses $B(1) = \{2,1,5,9,13,14\}$ and $B(2) = \{12,16,15,14,10,6,2,3,7,8,4\}$, where the numbers in the lists represent the processing element numbers arranged in order from the first (starting) processing element to the last (end). As an example, from Figure 3.3, communication between $PE[9]$ and $PE[13]$ on the bus $B(1)$ is made possible through the link $\{PE[9].s, PE[13].n\}$.

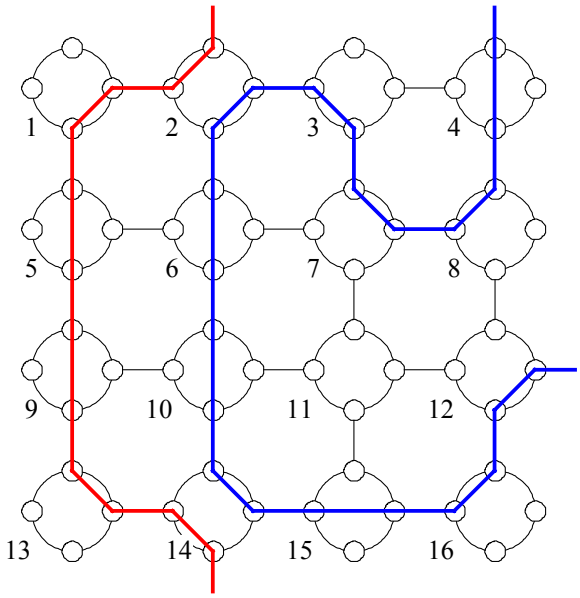


Fig.3.3: A 4 **x** 4 reconfigurable mesh network with two subbuses

The processing elements in a bus cooperate to solve a given problem by sending and receiving messages and data according to their controlling algorithm. A *positive direction* in a bus is defined as the direction from the first processing element to the last processing element, while the *negative direction* is the opposite. Note that the contents in the list of each bus at any given time t can change dynamically according to the current computational requirements.

3.3.2 Scheduling Model and Algorithm

In the model, *PE*[1] assumes the duty as the *controller* to supervise all activities performed by other processing elements in the network. This includes gathering information about the incoming tasks, updating the information about the currently executing tasks, managing the buses and locating the positions of the PEs for task assignments.

In our model, we assume the tasks to be nonpreemptive, independent and have no precedence relationship with other tasks. Hence, the computational model does not consider the communication cost incurred as a result of data transfers between tasks. We also assume the tasks to have no hard or soft executing deadlines. At time $t = 0$, the controller records Q_0 randomly arriving tasks, for $0 \le Q_0 \le Q$, and immediately places them in a FIFO queue, where Q is a predefined maximum number of tasks allowed. Each task $Task[i]$ is assigned a number i and a random length, denoted as $Task[i].length$. The controller selects q_0 connected PEs to form the bus $B(0)$ and assigns the Q_0 tasks to the q_0 PEs in $B(0)$. At this initial stage, the controller creates the bus list S to consist of a single bus $B(0)$, that is, $S = \{B(0)\}$. The PEs then start executing their assigned tasks, and their status are updated to "busy". Each PE broadcasts the information regarding its current execution status and the task it is executing to the controller, and the latter immediately updates this information.

This initial operation is repeated in the same way until the stopping time $t = StopTime$ is reached. At time t, Q_t random new tasks arrive and they are immediately placed in the FIFO queue. The queue line is created in such a way that every task will not miss its turn to be assigned to a PE. There are some Q_w tasks who failed to be assigned from the previous time slots, and these tasks are automatically in the front line. Hence, at any given time t, there are $Q_t + Q_w$ tasks in the queue, of which all Q_w tasks are in front of the Q_t tasks. In an attempt to accommodate these tasks, the controller forms m buses in the list $S = \{B(0), B(1),..., B(m)\}$. Each bus $B(j)$ has q_j connected PEs and this number may change according to the current processing requirements. The controller may add or delete the contents of each bus $B(j)$, depending on the overall state of the network. A PE in a bus $B(j)$ that has completed executing a task may be retained or removed from this bus, depending on the connectivity requirements for accommodating the tasks. The controller also checks the status of other PEs not in the list S. These PEs are not "busy" and may be added to the connecting buses in S. At the same time, some PEs may be transferred from one bus in S to another bus. In addition, the controller may also add or delete one or more buses in the list S to accommodate the same processing needs.

Finally, when the buses have been configured a total of q_t "free" PEs are then assigned to the q_t tasks in the front queue. When all the tasks have been completely executed, the controller compiles the information in its database to evaluate the mean arrival time λ_k, the mean executing time μ_k, and the mean response time R_k of each *PE*[*k*] in the network.

Our algorithm for scheduling the tasks dynamically on the reconfigurable mesh is summarised as follows:

```
At t = 0, the controller records Q_0 newly arriving tasks;
The controller selects q_0 connected PEs at random to form the bus B(0);
The Q_0 new tasks are assigned to the PEs in B(0);
The controller flags the assigned PEs in B(0) as "busy";
The controller creates the bus list S = {B(0)};
The controller updates the state information of the PEs in S = {B(0)};
for t = 1 to StopTime
    Q_t new tasks arrive while Q_w tasks still waiting;
    The controller places all the Q_t + Q_w tasks in the FIFO queue;
    The controller checks the state information of the PEs in B(j) of the
    list S = {B(0), B(1),..., B(m)} where B(j) ⊆ S:
    The controller checks the state information of the PEs not in S;
    The controller decides if the contents of B(j) need to change;
    The controller decides if the list S needs to change;
    The controller selects "free" PEs, assign them to the buses in S;
    The controller assigns q_t PEs to the q_t front tasks;
    The controller updates the state information of the PEs in S;
The controller evaluates λ_k, μ_k, and R_k of PE[k] in S;
```

3.4 Simulation and Analysis of Results

The simulation is performed on an Intel Pentium II personal computer. A C++ Windows-based simulation program called *Dynamic Scheduler on Reconfigurable Mesh* (DSRM) has been developed to simulate our model. DSRM assumes the tasks to have no partial orders, no communication dependence, no timing constraints and are nonpreemptive. Figure 3.4 shows a sample run of some randomly arriving tasks on a 4 **x** 4 network. In DSRM, every time tick t is a discrete event where between 0 to 10 randomly determined number of tasks are assumed to enter the queue waiting to be assigned to the PEs. For each task, its arrival time (randomly determined), length (randomly determined) and completion time, is displayed as a green or orange bar in the Gantt chart.

DSRM has some flexible features which allow a user-defined mesh network sizes of m **x** n, where $m, n = 1,2,...,64$. In addition, DSRM also displays the status of each processor in the network at time t as a square. A green square indicates the processor is busy as it has just been assigned a task, while a red square indicates the processor is also currently busy as it is still executing a previously assigned task. A black square indicates the processor is currently idle and, therefore, is ready for assignment. Figure 3.4 shows an instance of this discrete event at $t = 20$. $PE[3]$ is busy as it has just been assigned with Task 98, while $PE[7]$ is also busy as it is still executing Task 92. In contrast, $PE[11]$ is currently idle and is waiting for an assignment.

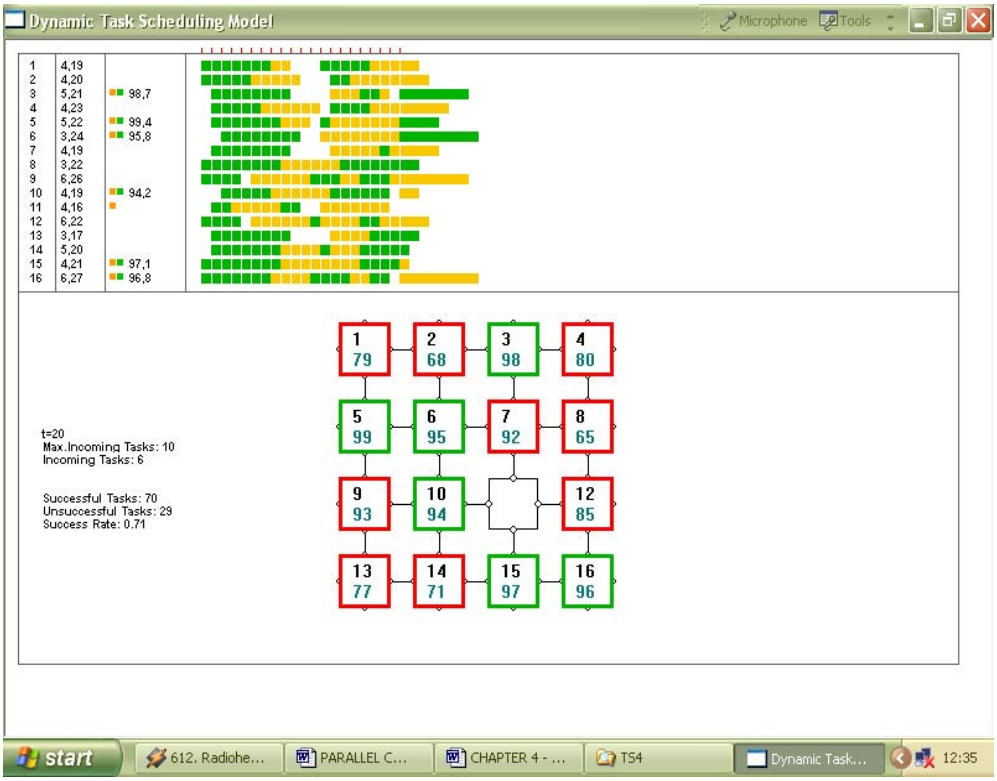


Fig.3.4: Sample run from DSRM

Results from a sample run of 209 successfully assigned tasks on a $4 \mathbf{x} 4$ network are shown in Table 3.1. Due to its dynamic nature, not all the tasks that arrive at time t managed to be assigned successfully on the limited number of processors. In this sample, 35 tasks failed to be assigned and this gives the overall success rate of 85.7%, which is reasonably good. In general, the overall success rate can be improved by controlling factors such as reducing the maximum number of arriving tasks at every time tick t and increasing the network size. In addition, it is possible to have a 100% success rate by bringing forward the unsuccessfully assigned tasks at time t to enter the queue at time $t+1$, $t+2$ and so on. These factors normally impose some timing constraints on the tasks, such as the execution deadline, and are presently not supported in DSRM.

The results from Table 3.1 show a fairly good distribution of tasks on the processors with a mean of 13.0625, with $PE[1]$ having the highest number of tasks (17), while $PE[5]$ has the

lowest assignment (9). The standard deviation is 2.3310, while the overall mean response time is 1.880. The tasks have a total execution time of 824 time units, with a mean of 51.5 and a standard deviation of 5.1720 on each processor. The table also shows the performances of each processor in the network, in terms of its mean arrival time, mean service time and mean response time, which describes a reasonably good distribution.

Table 3.1: Sample run of 209 successful randomly generated tasks on 16 PEs

PE	No. of Tasks	Total Exec. Time	Mean Arrival Time	Mean Service Time	Mean Response Time
1	17	45	0.261538	0.377778	8.60294
2	16	53	0.246154	0.301887	17.9427
3	12	60	0.184615	0.2	65
4	11	45	0.169231	0.244444	13.2955
5	9	48	0.138462	0.1875	20.3922
6	12	50	0.184615	0.24	18.0556
7	10	54	0.153846	0.185185	31.9091
8	16	52	0.246154	0.307692	16.25
9	12	54	0.184615	0.222222	26.5909
10	11	44	0.169231	0.25	12.381
11	14	50	0.215385	0.28	15.4762
12	15	52	0.230769	0.288462	17.3333
13	15	60	0.230769	0.25	52
14	13	58	0.2	0.224138	41.4286
15	11	44	0.169231	0.25	12.381
16	15	55	0.230769	0.272727	23.8333
Total	209	824			
Mean	13.0625	51.5			1.880
Std.Dev.	2.3310	5.1720			

3.5 Summary and Conclusion

This chapter describes dynamic task scheduling model implemented on the reconfigurable mesh computing model. The model is illustrated through our simulation program called *Dynamic Simulator on Reconfigurable Mesh* (DSRM) which maps a randomly generated number of tasks onto a network of m **x** n processors at every unit time t based on our scheduling algorithm. DSRM produces reasonably good load balancing results with a high rate of successful assigned tasks, as demonstrated in the sample run.

DSRM considers the tasks to have no partial orders, no communication dependence, no timing constraints and are nonpreemptive. These important factors will be considered in our future work as they are necessary in order for the model to be able to support many real-time and discrete-event requirements.

CHAPTER 4

RMESH MODEL FOR THE EDGE DETECTION PROBLEM

4.1 Introduction

This chapter presents our work in detecting the edges of an image using the reconfigurable mesh network. Edge detection is an important component of image processing which involves massive computations. Fast computers and good algorithms are some of the requirements in image processing. Due to its dynamic structure, the reconfigurable mesh computing model has attracted researchers on problems that require fast executions. These include numerically-intensive applications in computational geometry (Olariu *et. al*, 1994), computer vision and image processing (Olariu *et. al*, 1995) and algorithm designs (Nakano and Olariu, 1998).

The chapter is organized into four sections. Section 4.1 is the introduction. In Section 4.2, we discuss the edge detection problem and some methods for solving this problem. The reconfigurable mesh computing platform and model is explained in Section 4.3. In Section 4.4, we present our parallel Laplacian algorithm on the reconfigurable mesh network for solving the edge detection problem. Finally, Section 4.5 is the summary and conclusion.

4.2 Edge Detection Problem

The *edges* of an image form a separation line between pixels of the low intensity and the high intensity. *Edge detection* is a technique of getting this boundary line which holds the key to other image processing requirements, such as object recognition, image segmentation, image enhancement and image manipulation. Through edge detection, pixels can grouped according their variation in grey level or colour values based on some predefined threshold value. This information is vital to segmenting the image into two or more regions so that objects in the image can be detected or manipulated in ways appropriate to the problem.

Edge detection techniques aim to locate the edge pixels that form the objects in an image, minus the noise. Three main steps in edge detection are noise reduction, edge enhancement and edge localization. Noise reduction involves the removal of some unwanted noise pixels that sometime overshadow the real image. In edge enhancement, a filter is applied that responds strongly at edges of the image and weakly elsewhere, so that the edges may be identified as local maxima in the filter's output. Edge localization is the final step that separates the local maxima caused by the edges or by the noise.

The Laplacian edge detection method is a second order convolution that measures the local slopes of x and y of an image $f(x, y)$ (Effold, 2000). For an image of size q x r, the intensity of a pixel at coordinate `(i,j)` is represented in discrete form as f_{ij}, where $i = 0,1,2,..., r-1$ and $j = 0,1,2,..., q-1$. The Laplacian of this image is defined as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{4.1}$$

It follows that

$$\nabla^2 f = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2}$$

Assuming $\Delta x = \Delta y = 1$

$$\nabla^2 f = f_{i+1,j} - 4f_{i,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} \tag{4.2}$$

and this produces the x high pass filter [4]:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.3}$$

Similarly, the y high pass filter is given by the matrix [4]:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.4}$$

The sequential Laplacian method for detecting the edges of an image is given as follows:

```
/* Sequential Laplacian algorithm */
Let z = number of edges;
Let (xE, yE) be the x and y output, respectively;
Set the edge threshold ET be a constant;
Set (xH, yH) as the left-hand corner pixel of the output image;
for j=0 to q
   for i=0 to r
        Set f_ij = pixel value at (i,j);
Set z = 0;
for j=1 to q-1
   for i=1 to r-1
        xE = abs(f_{i,j+1} - f_{i-1,j} + 4f_{i,j} - f_{i+1,j} - f_{i,j-1});
        yE = abs(-f_{i-1,j+1} - f_{i,j+1} - f_{i+1,j+1} - f_{i-1,j} + 8f_{i,j} - f_{i+1,j} - f_{i-1,j-1} - f_{i,j-1} - f_{i+1,j-1});
        Set xyE = xE + yE;
        if xyE ≥ ET
             z++;
             Set E[z].x = xH +i and E[z].y = xH +j;
             Plot the edge pixel at (E[z].x, E[z].y);
```

4.3 Reconfigurable Mesh Computing Model

Our computing platform consists of a network of 16 processing elements arranged in reconfigurable meshes. A suitable realization for this model is the message-passing transputer-based system where each transputer represents a processing element with a processing element and a memory module each, and has communication links with other transputers.

Figure 4.2 shows a 4 x 4 network of 16 processing elements, $PE[i, j]$, for the rows $i = 0,1,2,3$ and columns $j = 0,1,2,3$. Each processing element in the network is capable of executing some arithmetic and logic operations. In addition, each processing element has some memory and four ports for communication with other PEs, denoted as $PE[i, j].n$, $PE[i, j].s$, $PE[i, j].e$ and $PE[i, j].w$, which represent the north, south, east and west links respectively. These ports can be dynamically connected in pairs to suit some computational needs.

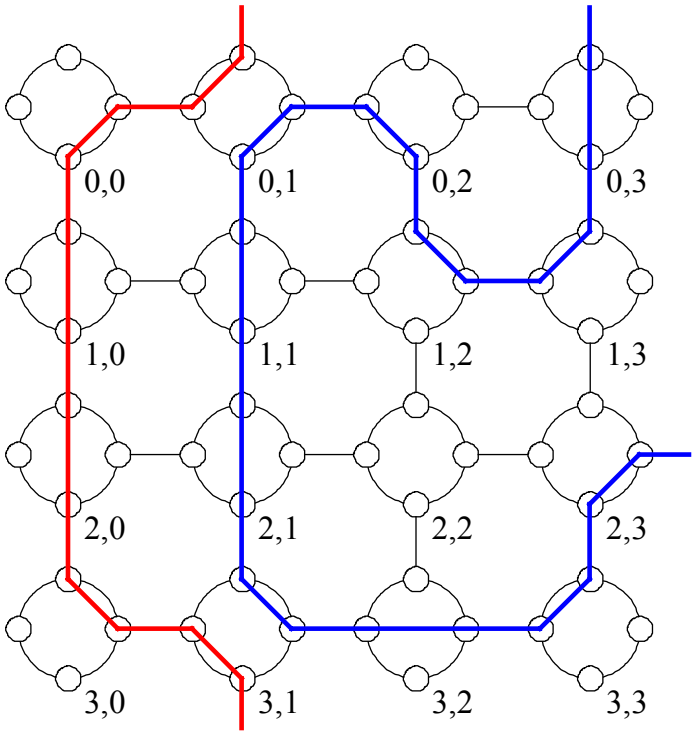


Fig. 4.1: A 4 x 4 reconfigurable mesh network with two subbuses

Communication between processing elements in the reconfigurable mesh can be configured dynamically in one or more buses. A bus is a doubly-linked list of processing elements, with every processing element on the bus being aware of its immediate neighbors. A bus begins in a processing element, pass through a series of processing elements and ends in another processing element. A bus that passes through all the processing elements in the network is called the *global bus*, otherwise it is called a *local bus.* Figure 4.2 shows two local buses $B(1) = \{2,1,5,9,13,14\}$ and $B(2) = \{12,16,15,14,10,6,2,3,7,8,4\}$, where the numbers in the lists represent the processing element numbers arranged in order from the first (starting) processing element to the last (end). As an example from the figure, communication between $PE[2,0]$ and $PE[3,0]$ on the bus $B(1)$ is denotated as $\{PE[2,0].s, PE[3,0].n\}$.

The processing elements in a bus cooperate to solve a given problem by sending and receiving messages and data according to their controlling algorithm. A *positive direction* in a bus is defined as the direction from the first processing element to the last processing element, while the *negative direction* is the opposite. Note that the contents in the list of each bus at any given time t can change dynamically according to the current computational requirements.

4.4 RM Model for Detecting the Edges

Our computing model is based on a n x n reconfigurable mesh network. As the model resembles a rectangular array, the computing platform is suitable for mapping the pixels of a q x r image directly for fast executions. For this purpose, we assume $n < q$ and $n < r$.

The Laplacian convolution of a q x r image involves scanning the n x n portion of the image beginning from the top left-hand corner of the image to the right, and continues downward continuously until the bottom right-hand corner is reached. This windowing process computes the second derivatives of f with respect to x and y, given as xE and yE in Equation (4.1), respectively. The convolution output at `(i,j)` is then given as $xyE = xE + yE$. This value is then compared to the edge threshold ET: if $xyE \geq ET$, an edge is present at location

$(E[z].x, E[z].y)$, where z is the edge number, otherwise the pixel is not an edge. Some initial assignments: f_{ij} is the intensity of the pixel at (i,j), ET is the edge threshold and (xH, yH) indicates the home coordinate where the binary edge image needs to be constructed.

It follows that our parallel algorithm using the RM model is summarized as follows:

```
/* Parallel Laplacian algorithm using a n x n reconfigurable mesh */
Let (xE, yE) be the x and y output, respectively;
Set (xH, yH) as the left-hand corner pixel of the output image;
Let the intensity at pixel (i,j) be f_ij;
At PE[0,0], set the number of edges z = 0;
At PE[0,0], set the edge threshold ET to be a constant;
PE[0,0] broadcasts ET southbound to PE[0,k], for k=1,2,…, n;
PE[0,k] broadcasts ET westbound to PE[h,k], for h=1,2,…, n;
for u=0 to q_4 step n, where
   for v=0 to r_4 step n, where r_4 = n⌊r/n⌋
      par j=u to u+n
         par i=v to v+n
            Set h = 1 + i% n;
            Set k = 1 + j% n;
            PE[h,k] evaluates xE = abs(f_{i,j+1} − f_{i−1,j} + 4f_{i,j} − f_{i+1,j} − f_{i,j−1}) and
        yE = abs(−f_{i−1,j+1} − f_{i,j+1} − f_{i+1,j+1} − f_{i−1,j} + 8f_{i,j} − f_{i+1,j} − f_{i−1,j−1} − f_{i,j−1} − f_{i+1,j−1});
            PE[h,k] evaluates xyE = xE + yE;
            if xyE ≥ ET at PE[h,k]
               PE[h,k] broadcasts a positive flag to PE[0,0];
               At PE[0,0], set z ← z+1 with the positive flag;
               At PE[h,k], set E[z].x = xH +i and E[z].y = xH +j;
               At PE[h,k], plot the edge pixel at (E[z].x, E[z].y);
```

As can be seen, the above algorithm has the complexity of $O(qr/n^2)$, against the sequential complexity of $O(qr)$. Finally, a C++ program to simulate the above algorithm has been developed. The program detects edges of up to 800×600 bitmap images, to produce their corresponding binary images.

4.5 Summary and Conclusion

We have presented a parallel Laplacian method using the reconfigurable mesh network for detecting the edges of an image . The algorithm is implemented in a C++ program that simulates n x n networks of various sizes to support up to 800 x 600 bitmap images. The method has the complexity of $O(qr / n^2)$.

CHAPTER 5

SINGLE-ROW ROUTING USING THE ENHANCED SIMULATED ANNEALING TECHNIQUE

5.1 Introduction

A typical VLSI design involves extensive conductor routings which make all the necessary wirings and interconnections between the PCB modules, such as pins, vias, and backplanes. In very large systems, the number of interconnections between the microscopic components in the circuitry may exceed thousands or millions. Therefore, the need to optimize wire routing and interconnection in the circuit is crucial for efficient design. Hence, various routing techniques such as single-row routing, maze routing, line probe routing, channel routing, cellular routing and river routing have been applied to help in the designs.

In So (1974), a divide-and-conquer approach has been proposed to deal with the complicated wiring problem in VLSI circuit design. The method begins with a systematic decomposition of the general multilayer routing problem into a number of independent single layer and single-row routing problems. This approach defines single-row routing problems for every horizontal and vertical line of

points in the original problem. The solutions of these sub-problems are then combined to contribute towards the overall solution to the original problem. Single-row routing (SRR) is a combinatorial optimization problem that finds its application in the design of VLSI multi-layer printed circuit boards (PCBs). The main objective in the single-row routing problem is to obtain a realization from the given routing that minimizes congestion on both the upper and lower streets of the circuit.

Single row routing problem has been shown to be NP-complete with large number of interacting degrees of freedom (Raghavan and Sahni, 1983). Most solutions to the problem have been expressed in the form of heuristic algorithms based on graph theory (as in Deogun and Sherwani, 1988), exhaustive search (as in Tarng *et al,* 1984) and greedy algorithms (as in Du and Liu, 1987). In Salleh and Zomaya (1999), a simulated annealing model called SRR-7 (*Single-Row Routing Model 7*) was introduced for solving the problem with the objective of minimizing both the street congestion (Q) and the number of doglegs (D). The model is based on an energy function E as a collective set representing both Q and D. Since the two parameters are allowed to vary freely during the annealing steps, the energy may, in some cases, produce optimum solution in one while ignoring the other.

In this chapter, we further improve on our earlier simulated annealing technique by expressing the energy as a function of one parameter. This process can be achieved by pivoting the other parameter to values not higher than its present value. The new approach is called ESSR (*Enhanced Simulated annealing for Single-row Routing*). In addition, ESSR involves the simultaneous swappings of all the nets in any single iteration. This approach has the effect of a faster convergence to the global minimum. The chapter is organized as follows: Section 5.2 is the problem statement, Section 5.3 discusses previous methods for solving the single-row routing problem, Section 5.4 is on our model, Section 5.5 presents the experimental results and analysis, while Section 5.6 is the conclusion.

5.2 Problem Background

5.2.1 Problem Formulation

In the single-row routing problem (Raghavan and Sahni, 1983), we are given a set of n evenly spaced terminals (pins or vias) $V=\{v_j\}, j=1,2,...,n$ arranged horizontally from left to right in a single row called

the *node axis*. The problem is to construct nets in the list $L = \{N_1, N_2, ..., N_m\}$ from the intervals $I = \{I_1, I_2, ..., I_m\}$. Each of these intervals is formed from a pair of two (or sometimes more) terminals through non-intersecting vertical and horizontal lines. The nets are to be drawn from left to right and the reverse direction is not allowed. The terminals for a given net are also called the net *touch points*. Physically, each net represents a conductor path for its terminals to communicate. Each path joining the terminals is called a *track*. An *interval* $I_i = (b, e)$ is the horizontal range between two terminals v_b and v_e that makes up the net N_i. A *unit interval* (a,a+1) is the interval between two successive terminals v_a and v_{a+1}.

The area above the node axis is called the *upper street*, while that below is the *lower street*. The number of horizontal tracks in the upper and lower streets are called the *upper street congestion* C_u and *lower street congestion* C_l respectively. The street congestion Q of a realization is defined as the maximum of its upper and lower street congestions, that is, $Q = \max(C_u, C_l)$. The congestion of an interval (b,e) can also be expressed as the *density* ρ, defined as the number of nets covering that interval.

Each terminal v_j has a *cut number* c_j, defined as the number of horizontal tracks a vertical line drawn through that point cuts. The nets cut by the vertical line are termed as the nets that *cover* the terminal. The nets are said to *cover from above* (*below*) if they lie above (below) the terminal. It can be shown that the street congestion given by $Q = \max(C_u, C_l)$ can also be expressed as (Raghavan and Sahni, 1983):

$$Q = \max\{\text{number of nets covering a terminal}\}. \qquad (5.1)$$

Our earlier algorithm (Salleh and Zomaya, 1999) evaluates the street congestion Q based on equation (5.1), as follows:

```
Algorithm SRR_CONGESTION
begin
     Set Q = 0;
     for j = 1 to J, where J = total number of terminals
        if Q < max(number of net covers from above on v_j)
            Q = number of net covers from above on v_j;
        endif;
        if Q < max(number of net covers from below on v_j);
            Q = number of net covers from below on v_j;
        endif;
     endfor;
end;
```

The *upper* (*lower*) *cut number* c_{ju} (c_{jl}) is the number of nets cut by the vertical line through v_j from above (below). The cut number q_i of net N_i is the maximum of the cut numbers of the net left and right terminals. For the interval (b,e), c_b and c_e are the net total beginning (left) and end (right) cut numbers, respectively. The vertical position of a terminal v_j in a net ordering is called *position*, denoted by pos (j).

A vertical line crossing the node axis in a given realization is called a *dogleg*. A dogleg is necessary since it represents the sudden detour of a track in order to avoid crossing another track. The presence of doglegs, however, increases the system overhead as they add to the circuit complexity. Therefore, minimizing the number of doglegs D is another important circuit design objective as it contributes towards the design of a more compact realization.

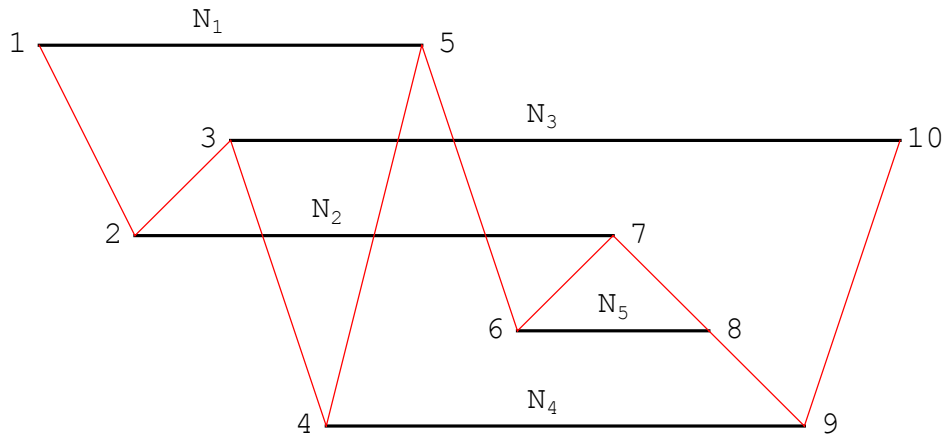


Fig. 5.1: Net ordering $L = \{N_1, N_3, N_2, N_5, N_4\}$ with the reference line

Figure 5.1 shows five nets in the order $L = \{N_1, N_3, N_2, N_5, N_4\}$ formed from the following intervals: $N_1 = (1,5)$, $N_2 = (2,7)$, $N_3 = (3,10)$, $N_4 = (4,9)$ and $N_5 = (6,8)$. The line joining the vertices successively from left to right is called *reference line*. The reference line is important in the design as it gives a preview of the graphical realization. It can be seen, for example, from the figure that v_6 is covered from above by N_2 and N_3, and from below by N_4. The net ordering in the figure gives a street congestion value $Q = 3$, as v_5 has 3 nets covering from below ($C_l = 3$) and v_4 has 3 nets covering from above (C_u=3). In the figure, a dogleg is marked from the crossing of the reference line on any interval. It is easy to verify that the number of doglegs D is 3 with a dogleg present in each of of the intervals (3,4), (4,5) and (5,6).

The following algorithm (Salleh and Zomaya, 1999) outlines a method to determine the number of doglegs d_j in the interval $(j, j+1)$ using the horizontal position $pos(i)$ of vertex i:

```
Algorithm SRR_NDOGS
begin
Set d_j = 0;
In the given interval (j, j+1):
    for i = 1 to m
        if N_i covers (j, j+1)
            if pos(j) ≤ pos(i) ≤ pos(j+1)
                d_j ← d_j + 1;
            endif;
            if pos(j) ≥ pos(i) ≥ pos(j+1)
                d_j ← d_j + 1;
            endif;
        endif;
    endfor;
end;
```

In Olariu and Zomaya (1996), a graph theoretic technique has been applied to produce the maximal interlocking set from a set of net intervals. The results have been used to obtain a time- and cost-optimal realization without doglegs with $O(\log n)$ time complexity using $\frac{n}{\log n}$ processors in the CREW-PRAM model. It has been shown that the realization is possible only if the corresponding overlap graph is bipartite.

Vertices that are local maxima with respect to the reference line are called *peaks*, while those that are local minima are *valleys*. In the above figure, v_3, v_5 and v_7 are peaks, while v_2, v_4 and v_6 are valleys. It is also noted that N_2, for example, has 4 *segments* formed from the intersection between the reference line and the interval (2,7). The segments of a net N_i are labeled as $N_{i,r}$, for i=4 and r=1,2,3,4. The number of segments in a given interval (b,e) is determined as follows (Salleh and Zomaya, 1999):

No. of segments = No. of doglegs on (b,e) +1 (5.2)

The *height* $h_{i,r}$ of the segment $N_{i,r}$ is defined as the vertical unit distance of that segment from the node axis and is determined as follows:

$$h_{i,r} = \text{Maximum number of nets covering a peak or a valley in } N_{i,r} \qquad (5.3)$$

Algorithm `SRR_SegmentHeight` (Salleh and Zomaya, 1999) below outlines a method to determine the height $h_{i,r}$ of the segment $N_{i,r}$:

```
Algorithm SRR_SegmentHeight
begin
    Given a segment N_{i,r}:
    Set h_{i,r} = 0;
    if the number of peaks > the number of valleys
        for w = 1 to w_r, where w_r = number of peaks in N_{i,r}
            if h_{i,r} < number of nets covering v_w from N_{i,r}
                h_{i,r} = number of nets covering v_w from N_{i,r};
            endif;
            Update h_{i,r} ← h_{i,r} − 1;
        endfor;
    endif;
    if the number of peaks < the number of valleys
        for w = 1 to w_r, where w_r = number of valleys in N_{i,r}
            if h_{i,r} < number of nets covering v_w from N_{i,r}
                h_{i,r} = number of nets covering v_w from N_{i,r};
            endif;
            Update h_{i,r} ← h_{i,r} + 1;
        endfor;
    endif;
end;
```

A height with a "+" value means the segment is in the upper street, while the "-" value means it is in the lower street. It can been seen from the figure that the heights of segments $N_{2,1}$, $N_{2,2}$, $N_{2,3}$ and $N_{2,4}$ are -1, +1, -2 and +1, respectively. Also, the height of N_4 is -3 as 3 nets cover the valley v_5 and 2 nets covering v_7. It is also easy to verify that the maximum height of the segments in an interval gives the street congestion of that net.

5.2.2 Necessary and Sufficient Conditions

A *realization* or routing is said to be achieved when all the nets are successfully drawn for the given terminals, satisfying all the following conditions (Ting *et al.*, 1976):

(1) $N_i \bigcap N_j = \phi$, for $i \neq j$

(2) $\bigcup N_i = \{1,2,\ldots,n\}$

(3) Each path is made up of horizontal and vertical segments only.

(4) The paths do not cross.

(5) The path movement is in forward direction, and backward move is not allowed.

The optimality of SRR is very much related to the order of the nets in a given list, *L*. Hence, SRR produces solutions in a similar fashion to the travelling salesman problem. For example, Figure 5.2 shows the graphical realization corresponding to the net ordering $L = \{N_1, N_3, N_2, N_5, N_4\}$ from Figure 5.1.

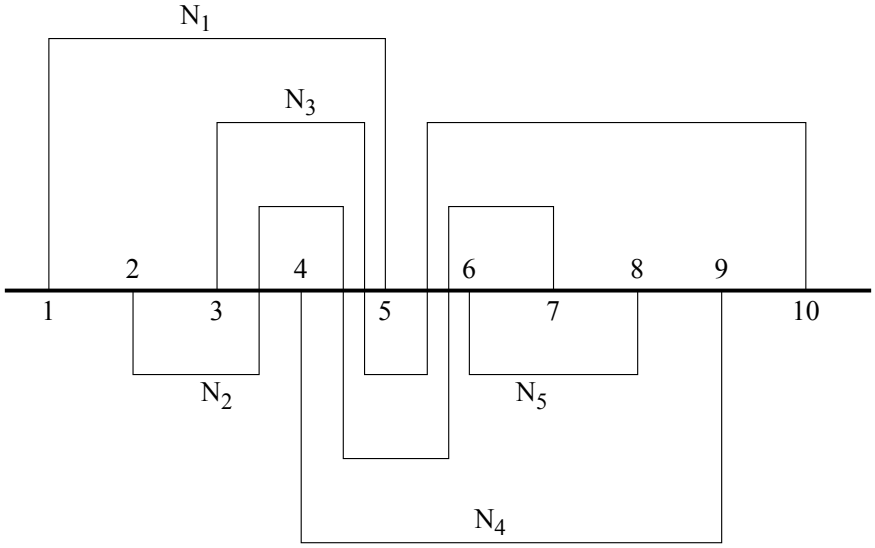


Fig. 5.2: Realization from the ordering $L = \{N_1, N_3, N_2, N_5, N_4\}$

The following relationships for j=1,2,...,m are obtained from (Kuh *et. al*, 1979):

$$C_u = \max_j(c_{ju}) \tag{5.4a}$$

$$C_l = \max_j(c_{jl}) \tag{5.4b}$$

$$Q = \max(C_u, C_l) = \max_j\{\max(c_{ju}, c_{jl})\} \tag{5.5}$$

An optimal realization is a realization that represents the best permutation of nets which satisfies a number of optimality conditions. Several possible performance metrics defining optimal realizations are stated as follows:

(1) Realization with minimum congestion on both streets. Our problem of finding an optimal realization amounts to a massive search for the best ordering of m nets among the $m!/2$ permutations that produce $Q_{\min} = \min(Q)$. The objective function is, therefore Q, and our problem is to find $Q_{\min} = \min(Q)$ which is the global minimum.

(2) Realization with minimum number of doglegs. The objective function in this case is the number of doglegs D generated in the realization, and our objective is to find $D_{\min} = \min(D)$. With a small number of doglegs, the circuit physical size is reduced and this helps in making it more compact.

(3) Realization with a bounded number of doglegs d_0 in every unit interval. Since each interval in the node axis $(i, i+1)$ have equal spacing, it is expected that the distribution of doglegs among them, $d_{i,i+1}$, will vary. Some intervals will have too many doglegs, while some others may not have any. This adds constraints to its design since an unbalanced distribution may cause excessive conductor wiring in some area. Therefore, it is desired that the interval crossings are bounded at every interval, and that the doglegs are evenly distributed at all intervals.

(4) Realization that minimizes the maximum number of doglegs in every net. With a small number of doglegs in a net, the overall length of the track is shortened. This, in turn, reduces communication between the terminals and, therefore, improves the circuit performance.

(5) A combination of one or more of the above.

In general, it is difficult to determine an optimal realization for a given problem due to the large number of interacting variables in the problem, especially when the number of nets is large. A *feasible* realization, that is, the one that approximates the solution close to its optimal value, is accepted in many cases. The feasible solution must, however, satisfy the necessary and sufficient conditions stated in the following theorems:

Theorem 5.1 (Kuh *et al.*, 1979)**:** Let $q_m = \min_i(q_i)$ and $q_M = \max_i(q_i)$, then $Q_{\min} \geq \max\{q_m, q_t\}$, for $q_t = \lceil q_M / 2 \rceil$, and i=1,2,...,m, where m is the total number of nets.

This theorem asserts that the lower bound for a congestion is given by $Q_{\min} \geq \max\{q_m, q_t\}$ while the upper bound is $\max_i(q_i)$. However, the optimal solution $Q_{\min} = \max\{q_m, q_t\}$ is not guaranteed and it may be impossible to find especially when the size of the problem grows. The following two theorems describe this situation:

Theorem 5.2 (Kuh *et al.*, 1979): An optimal realization with street congestion $Q_{\min} = \lceil q_M / 2 \rceil$ exists if and only if there exists such an ordering that, for each v_j with $c_j = \lceil q_M / 2 \rceil + k$ where k=1,2,...,q_M-$\lceil q_M / 2 \rceil$, the net associated with v_j is covered from above and below by at least k nets.

Theorem 5.3 (Kuh *et al.*, 1979): An optimal realization with street congestion $Q_{\min} = \lceil q_M / 2 \rceil + p$ exists if and only if p is the least nonnegative integer for which the p-excess property holds. The p-excess property states that for each v_j the net associated with v_j is covered by at least k-p nets from above and below, given by $c_j = q_t + k$ where k=p+1,..., $q_M - \lceil q_M / 2 \rceil$.

5.3 Review of State of the Art

Due to their practical importance, single-row routing problem has been studied extensively. In Kuh *et al*. (1979), an algorithm based on a graph-theoretical interpretation of the problem produces optimal solutions. However, the method has the complexity $O(m!)$ for m nets, which is exponential. In Raghavan and Sahni (1983), an optimal solution is obtained when the number of tracks available on each street is known in advance. The heuristic method is also exponential in nature. Several other solutions using heuristics produced over the past few years also end up with exponential complexities. This development suggests that the problem is, in general, intractable and, therefore, is NP-complete.

In Tarng *et al.* (1984), the nets are arranged by placing those with lower cut number in outer rows and those with higher cut numbers to the middle. It also suggests that the cut number c_j of any terminal v_j is to be divided properly between c_{ju} and c_{jl} at those terminals where the cut number is larger than the net minimum cut number q_m.

Some terminologies are used to describe Tarng *et al.*'s heuristic. Given a net list $L=\{N_1,N_2,...,N_I\}$ and a division of L into two sublists L_1 and L_2 such that $L_1 \cap L_2=\phi$ and $L_1 \cup L_2=L$. The *internal cut number* of $N_i \in L_1$ in L with respect to L_1 is defined as the cut number of N_i in L_1. The *residual cut number* of N_i in L with respect to L_1 is defined as the cut number of N_i in $\{L_2 \cup N_i\}$.

In the implementation, the nets are first sorted according to their classes, followed by the internal cut numbers and finally by the residual cut numbers. Nets are said to belong to the same *class* if they have the same number of cut numbers. Nets in the same class are arranged in the descending internal cut numbers. If two nets belong to the same class and have the same internal cut numbers, then the one with larger residual cut number precedes the one with smaller residual cut number. The classification method produces a zoning list based on the cut numbers and further zoning based on virtual tracks. Example 1 below illustrates how the heuristic is implemented.

Example 1*:* Given a set of net intervals: N_1=(1,5), N_2=(2,6), N_3=(3,7), N_4=(4,12), N_5=(10,11), N_6=(9,15), N_7=(8,18), N_8=(13,16) and N_9=(14,17). We need to find the net ordering that produces the minimum street congestion.

Table 5.1: Classification of nets in Example 1

i	interval	q_i	class	int.cut no.	res. cut no.
N_5	(10,11)	3	L_0	2	1
N_9	(14,17)	3	L_0	2	1
N_1	(1,5)	3	L_0	1	2
N_4	(4,12)	3	L_0	1	2
N_6	(9,15)	3	L_0	1	2
N_2	(2,6)	2	L_1	1	1
N_3	(3,7)	2	L_1	1	1
N_8	(13,16)	2	L_1	0	2
N_7	(8,18)	1	L_2	0	1

Table 5.1 classifies the nets according to the cut number q_i, internal cut numbers and residual cut numbers, from the given intervals. Figures 5.3 shows the resulting net ordering and realization, respectively. The final net ordering from the algorithm is given as follows: $L = \{N_2, N_8, N_4, N_1, N_7, N_9, N_6, N_3, N_5\}$, with $Q = 2$ and $D = 5$. The realization is optimal in terms of minimum street congestion. By coincidence, the number of doglegs is also optimum although the algorithm does not take this criteria as a performance objective.

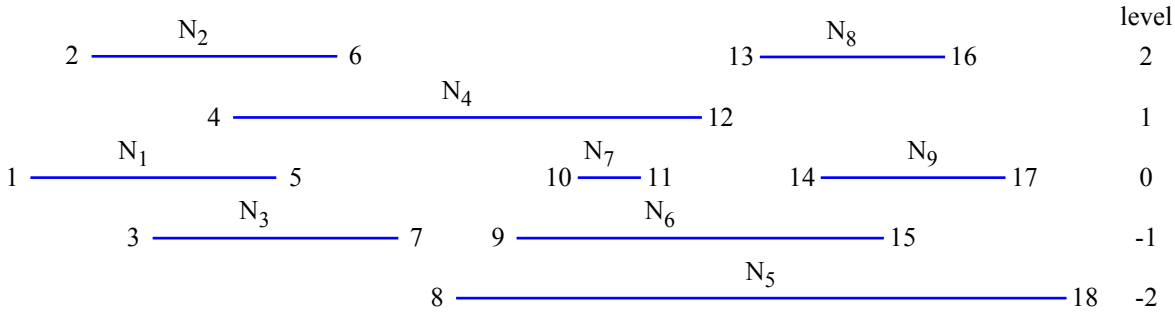


Fig. 5.3(a): Final net ordering based on virtual tracks

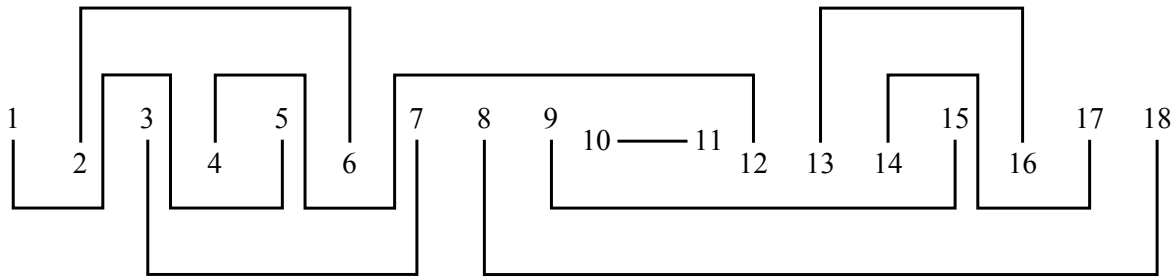


Fig. 5.3(b): Final realization with $Q = 2$ and $D = 5$

Tarng *et al.*'s heuristic has a number of shortcomings. It was demonstrated in Du and Liu (1987) that Tarng's heuristic generates optimal solutions in cases where all the given nets cover at least one common terminal. In cases where not all the nets cover a common terminal, the heuristic may not be optimal. Therefore, there is a need to further rearrange the nets using other net properties to improve on the results.

Du and Liu's heuristic considers the net grouping situation before sorting the nets according to their classes, internal cut numbers and residual cut numbers. A *group* is defined as the set of nets that cover more than one terminal. Therefore, in a given situation more than one group of nets maybe formed. Tarng *et al.*'s heuristic is optimal only if all the nets can be formed into one group. In contrast, Du and Liu's heuristic improves the computational complexity in the worst-case performance to $O(mn)$. In most other cases, the algorithm performs faster and generates better near-optimal solutions.

We briefly illustrate Du and Liu's heuristic using Example 1. Figure 5.4(a) shows the net position when the zoning considers net classification into groups. The heuristic produces the list $L = \{N_2, N_8, N_4, N_1, N_7, N_9, N_6, N_3, N_5\}$ with $Q = 2$ and $D = 6$, which is optimal in terms of minimum street congestion. Figure 5.4(b) shows the realization from this list.

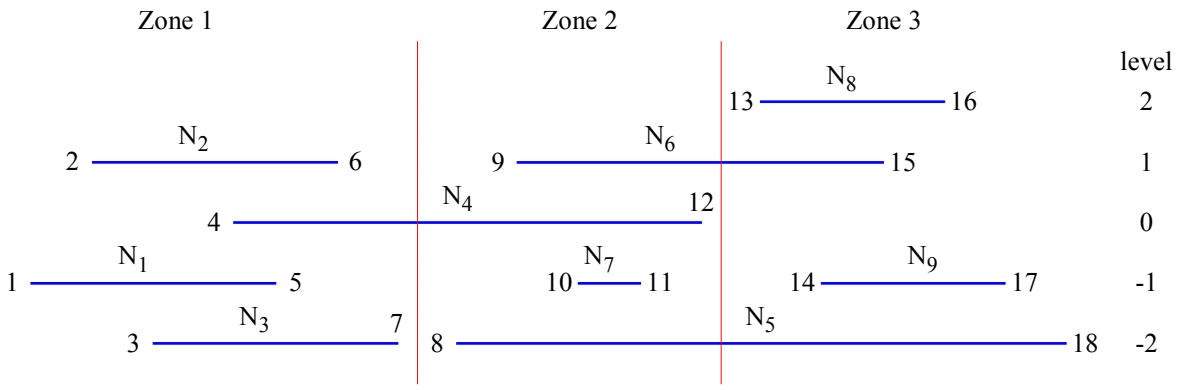


Fig. 5.4(a): Classification according to zones

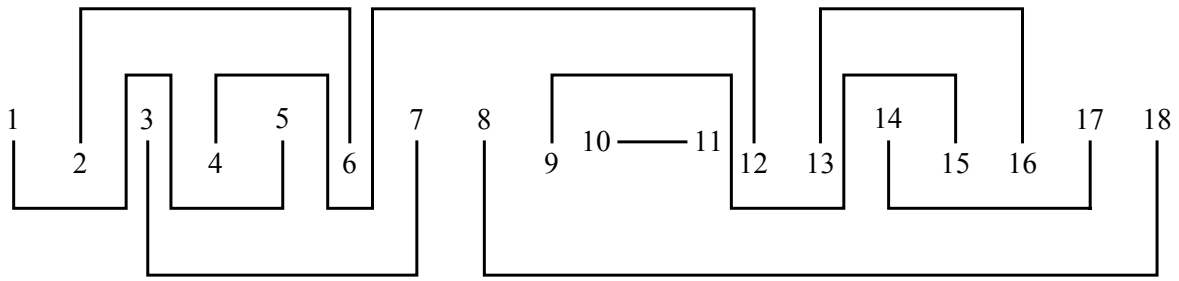


Fig. 5.4(b): Final realization with $Q = 2$ and $D = 6$

5.4 Enhanced Simulated Annealing Approach

Both heuristic approaches in Section 5.3 consider the minimum street congestion as the only performance objective. The realization may produce many intolerable doglegs although the objective of minimum street congestion is achieved. As mentioned earlier, the presence of doglegs may increase the track length, and this adds to the circuit complexity and overhead. Therefore, it is our objective in this section to find a realization that minimizes both the street congestion Q and the number of doglegs D.

Earlier in Salleh and Zomaya (1999), we presented a model for minimizing both Q and D using simulated annealing. It is possible to apply simulated annealing (SA) in VLSI design as demonstrated

by Rutenbar (1989). In his work, SA is used to design the chip layout and floorplanning. As discussed earlier, simulated annealing is a hill-climbing, gradient-descent technique that has the disadvantage of slow convergence to the solution. However, SA often produces good solutions that are comparable to other techniques. The SA method based on the Metropolis Algorithm (Aarts and Korst, 1989 and Kirkpatrick *et al.*, 1983) implements the Boltzmann distribution function as its energy minimization network. This probability function is known to have the mechanism to escape from getting trapped in a local minimum. Therefore, convergence is guaranteed although at the expense of long computation time. In addition, SA is easier to implement as the objective function does not have to be in an explicit functional representation.

5.4.1 Energy Formulation

Our objective in this subsection is to obtain a realization that minimizes both the street congestion Q and the number of doglegs D. However, this objective is very difficult to achieve as the two components are separate but dependent entities. While having one component minimized, the other tends to show some resistance to minimization. We solved this difficulty by expressing both components as a single energy function. To express this requirement, the energy in a given net ordering is expressed as the total length of all the tracks.

Since the horizontal length of each interval is fixed, this energy depends on the vertical length of the tracks. That is, the total energy E_L of nets in a given list is directly proportional to the sum of the energy of net segments, given as follows (Salleh and Zomaya, 1999),:

$$E_L = \sum_{i=1}^{m} \sum_{j=1}^{m_i} \left| h_{i,j} \right| \tag{5.6}$$

where m_i is the number of segments in the net N_i, for i=1,2,...,m. In this equation, $\left| h_{i,j} \right|$ is the energy of segment j of net i, which is the absolute value of the height $h_{i,j}$ of that segment relative to the node axis.

Figure 5.5 shows a list $L = \{N_6, N_4, N_9, N_2, N_5, N_8, N_7, N_1, N_3\}$ from Example 1. Using Equation (5.6), it is easy to verify that $E_L = 33$.

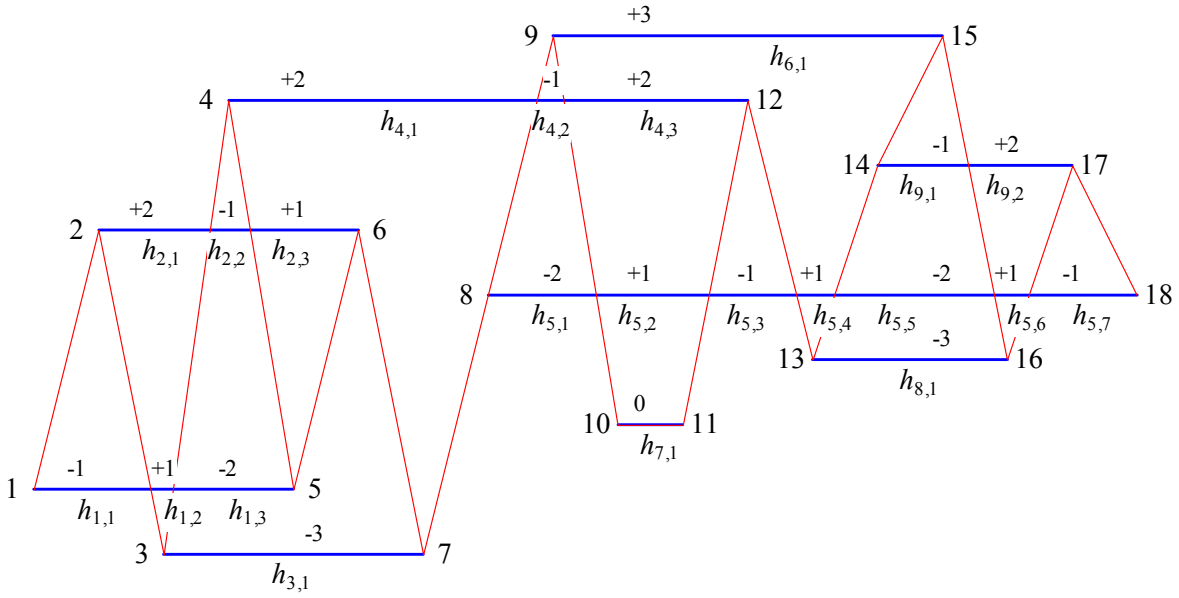


Fig. 5.5: An ordering showing net segments and their heights

5.4.2 ESSR Model

Simulated annealing is an iterative improvement process of local search for the global minimum of a given energy function. This method is based on the simple and natural technique of trial and error. SA requires the definition of a solution space, a cost function and a set of moves that can be used to modify the solution. Through the iterative improvement method, one starts with an initial solution y_0 and compares this solution with its neighbors. A solution y' is said to be a *neighbor* of a solution y if y' can be obtained from y via one of the moves. The neighbors of y_0 are examined until a neighborhood solution y with a lower cost is discovered. In this case, y becomes the new solution and the process is continued by examining the neighbors of the new solution. The algorithm terminates when it arrives at a solution which has no neighboring solution with a lower cost.

In our previous solution (Salleh and Zomaya, 1999), a pertubation is performed to examine the neighbors by swapping the position of two nets at random. The resulting change in energy ΔE is then evaluated. If the energy is reduced, or $\Delta E \leq 0$, the new configuration is accepted as the starting point for the next move. However, if the energy increases, or $\Delta E > 0$, the move generates the Boltzmann probability, given by $\Pr\{acceptance\} = e^{-\frac{\Delta E}{T}}$. The move is accepted if this probability is greater than a threshold probability of acceptance ε and rejected otherwise. The value of ε is proportional to the rate of acceptance of rejection. With a higher value, the number of moves accepted for $\Delta E > 0$ are reduced and the same rule applies vice versa.

The above technique works well in producing optimal solutions in cases of up to 80 nets. However, the time taken for convergence may be very long in some cases. This problem may be caused by the fact that only one pair of nets are swapped at a given temperature. Furthermore, simulated annealing is a stochastic process which depends on the strict Boltzmann probability for convergence. We improve on the convergence through ESSR, which involves the movement of all nets rather than just one, at any given temperature.

We now describe how ESSR works. First, the parameters are initialized according to simulated annealing requirements. At the initial temperature $T = T_0$ (T_0 is assigned a value such as 100), the following parameters are given random values: the reducing parameter α (for example $\alpha = 0.9$) and the Boltzmann probability threshold ε (for example ε = 0.85). An initial net list L_0 is chosen at random and its energy E_0 is evaluated. A pair of nets, denoted as r, are then chosen and their positions are swapped to produce a new configuration L_r. We then evaluate the energy E_r, congestion Q_r and dogleg D_r. The energy difference $\Delta E = E_r - E_0$ is computed. The acceptance $(L_0 \leftarrow L_r)$ or rejection (no change in L_0) of this new configuration follows the same rule as above. We next choose another pair of nets for swapping different from the previous pair, and apply the same energy update criteria to determine if L_0 changes or not. The same process is repeated until all $\left\lfloor \frac{m}{2} \right\rfloor$ pairs of nets have been tested for the energy update. The final list L_0 may or may not change during these swapping processes.

The above procedure is repeated at each temperature $T_1, T_2, \ldots, T_c$ where $T_k \leftarrow \alpha\, T_{k-1}$, for $k = 1,2,...,c$ and T_c is the *critical temperature*, or the temperature at which the energy stabilizes. At each of these decreasing temperatures, the Boltzmann probability tends to decrease. This means there is a higher rejection rate. Hence, the acceptance procedure becomes stricter, to allow more stable solutions. At $T = T_c$, the energy E_c is at its global minimum and the individual from L_u and L_v with the lower energy then represents the desired net configuration.

We further enhance the convergence criteria in ESSR by preserving the street congestion value Q, while at the same time allowing the number of doglegs to decrease. This is necessary since the energy function E is a function of both Q and D. A minimum E contributes to minimum Q and D as a grouped object, but not necessarily on each of these variables individually. Therefore, by pivoting one parameter, for example Q, to some fixed value E can be made directly proportional to the other parameter D. At each temperature T_k the last street congestion value Q_k is maintained in such a way that changes to the net configuration are allowed only if the new value Q_r is lower or has the same value as Q_k, besides obeying the acceptance/rejection rule. However, at the following temperature T_{k+1}, Q_k is only allowed to change to the same value or to a lower value, according to its energy change.

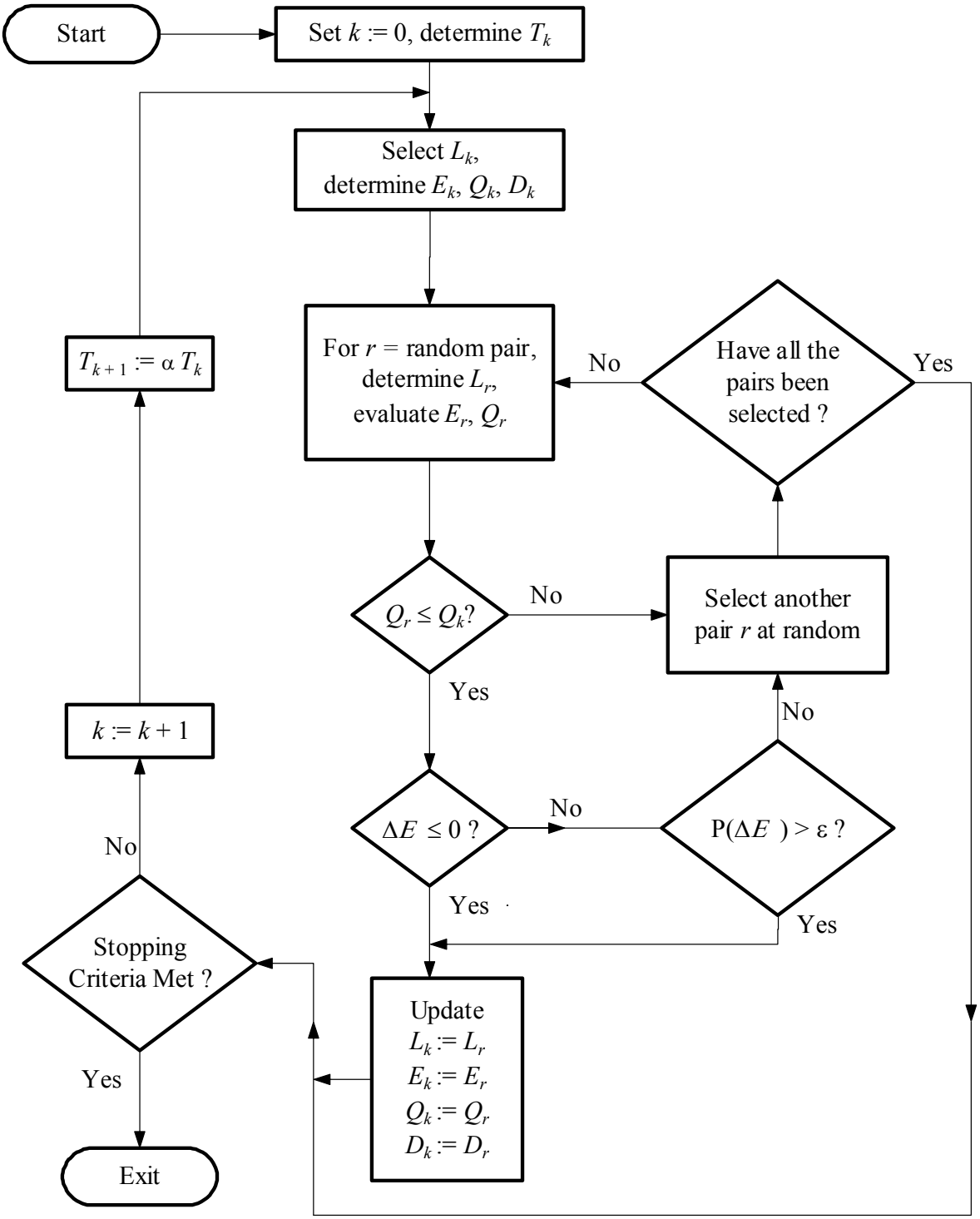


Fig. 5.6: ESSR flowchart

The whole steps in the implementation of ESSR is shown as a flowchart in Figure 5.6. In addition, Algorithm ESSR below summarizes our simulated annealing model for this problem:

```
Algorithm ESSR
(by pivoting Q)
begin
Select a suitable value for α ≈ 1, for 0 < α < 1;
Let k = 0 and select the starting temperature T_0;
Choose an initial list L_0;
Compute E_0 (using Equation 6) and Q_0 (using SRR_CONGESTION);
while T = T_k is in the cooling range
    for j = 1 to ⌊m/2⌋
        Select one pair of nets and swap their positions to form L_r;
        Evaluate the new energy E_r and congestion Q_r;
        if Q_r ≤ Q_k
            if (ΔE = E_r − E_k ≤ 0), or (if ΔE > 0 and exp(−ΔE/T_k) > ε )
                Update L_k ← L_r and E_k ← E_r;
                Evaluate Q_r and update Q_k ← Q_r;
            endif;
        endif;
    endfor;
    Test for the stopping criteria:
    if ΔE < δ after some small number of iterations
        Evaluate the final D_k using SRR_NDOGS;
        Obtain the final values E_k, Q_k, D_k and L_k;
        Obtain the realization from L_k;
        Stop and exit;
    else
        Update k ← k + 1 and T_k ← α T_{k−1};
    endif;
endwhile;
```

Example 2: This example illustrates ESSR using the nets in Example 1. We set the initial temperature $T_0 = 100$, the reducing parameter $\alpha = 0.95$ and the threshold probability of acceptance $\varepsilon = 0.95$. It can be verified that this net list results in $E_L = 33$. The iterative improvement process in ESSR reorders the nets to $L = \{N_2, N_8, N_9, N_4, N_5, N_6, N_7, N_1, N_3\}$ at equilibrium after 24 moves with the final net energy $E_L = 17$. Figure 7.7 shows the graph of the net energy against the accepted moves, while Table 7.2 tabulates the instances of the accepted moves with the Boltzmann probability.

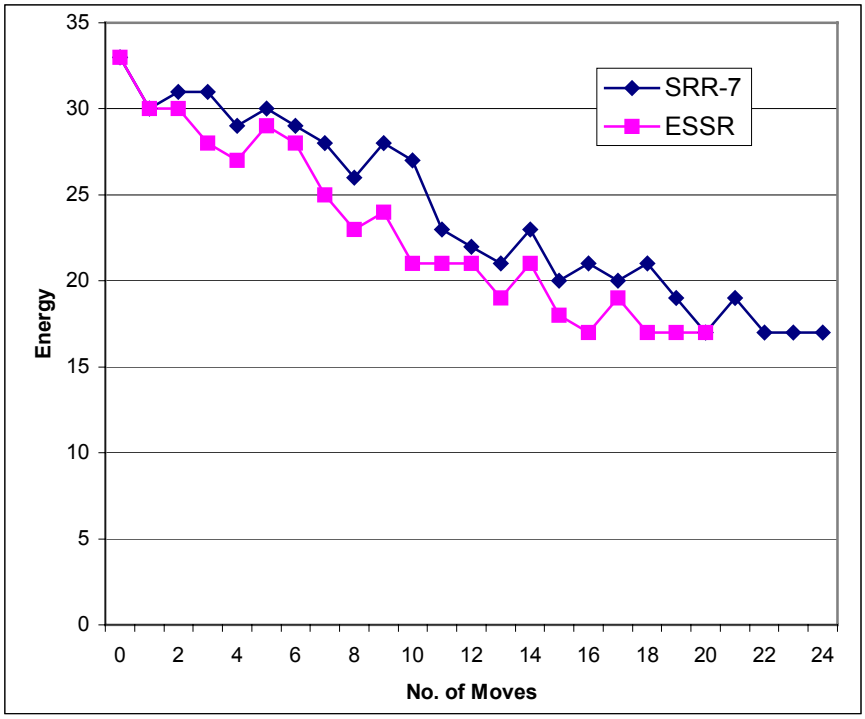


Fig. 5.7: Comparison between ESSR and SRR-7

The final net ordering yields $Q = 2$ and $D = 5$, which is optimal both in terms of minimum street congestion and minimum number of doglegs. Figure 5.8 shows the graphical realization of the nets at equilibrium.

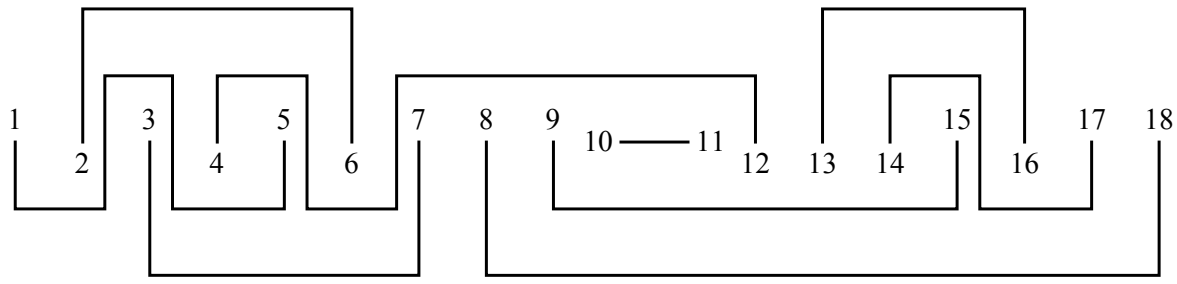


Fig. 5.8: Final net realization of Example 1 using ESSR

6.5 Experimental Results and Analysis

We tested our model on several net sizes and configurations. A program called ESRR.EXE, coded in C, has been developed to perform the simulations. The results are compared to our earlier model SSR-7 (Salleh and Zomaya, 1999), and to the heuristic approaches by Tarng *et al.* (1984), and Du and Liu (1987).

In all the cases, we use $T_0 = 100$ as the initial value for the thermostatic temperature *T*. This value is gradually reduced geometrically using Algorithm `SRR_SA`, with $\alpha = 0.95$. The threshold probability of acceptance is set at $\varepsilon = 0.9$.

Table 5.2: Comparison of experimental results

Data Set#	No. of Nets	*Q*,*D*			
		ESSR	SRR-7	Tarng *et al.*	Du and Liu
1	8	2,5	2,5	2,6	2,6
2	9	2,5	2,5	2,5	2,6
3	10	3,5	3,5	3,8	3,7
4	12	5,3	5,3	5,8	5,9
5	13	5,5	5,5	5,8	4,10
6	14	5,8	5,8	5,12	5,10
7	18 (set 18a)	6,9	6,10	5,14	5,10
8	18 (set 18b)	6,14	6,14	6,17	6,16
9	20 (set 20a)	6,13	6,13	6,19	5,18
10	20 (set 20b)	7,12	7,14	7,24	7,21
11	24	8,13	8,13	7,24	7,21
12	30	11,17	11,18	10,25	10,26
13	36	11,28	13,27	11,37	11,40
14	40 (set 40a)	15,25	16,28	16,34	16,43
15	40 (set 40b)	16,28	16,29	16,38	16,34
16	40 (set 40c)	16,32	16,35	not available	not available
17	50	23,155	23,161	not available	not available
18	80	24,304	28,312	not available	not available

Figure 5.9 shows a ESSR simulation of 20 nets using Model 9 from the above table. An initial list is first obtained at $T = 100$ to produce $E = 251$, $Q = 7$ and $D = 53$. Q is pivoted at this value or lower while E and D values are allowed to vary according to the annealing steps. Obviously, the energy drops almost proportionally as the number of doglegs during the simulation. The simulation produces $Q = 7$ until $T = 35.9$ where the value drops to 6, while $E = 88$ and $D = 17$. The energy stabilizes at $T = 8.5$ for $E = 82$, $Q = 6$ and $D = 13$, which are all minimum. The results obtained from this model are very optimum both in terms of minimum street congestion and minimum number of doglegs. The results obtained in other models, as tabulated in Table 5.2, also optimum in most cases.

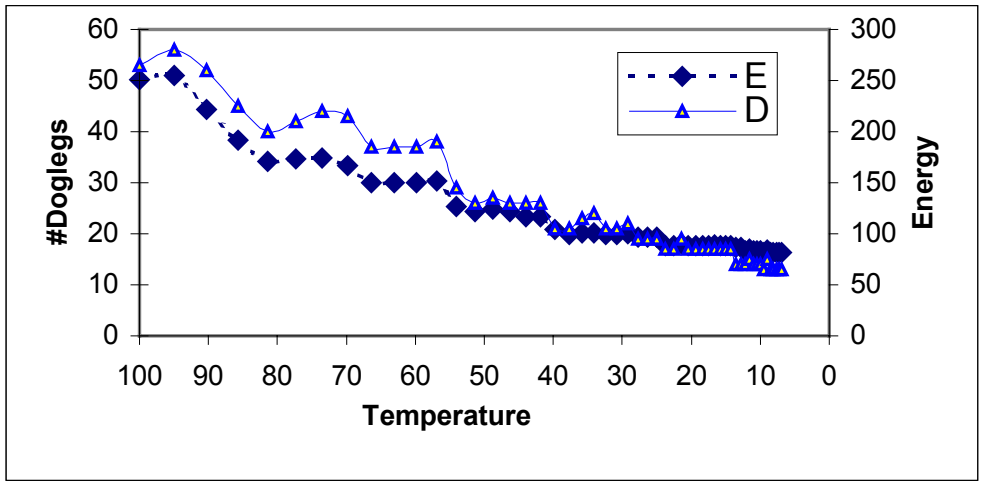


Fig. 5.9: Energy against the number of doglegs in ESSR by pivoting Q

Table 5.2 also suggests that ESSR performs better than the other three previous methods when the number of nets is large. This effect can be seen especially in cases of 40, 50 and 80 nets where the time taken for convergence to the equilibrium is longer. Both, the energy and the number of doglegs, drop drastically during the annealing steps. For example, in our 50-net set, the initial values are $E = 6{,}810$, $Q = 26$ and $D = 667$. At equilibrium these values converge to $E = 1{,}352$, $Q = 23$ and $D = 155$. Our 80-net set in Model 18 initially has $E = 11{,}792$, $Q = 26$ and $D = 1{,}206$ at $T = 100$. After 256 steps the energy stabilizes at $E = 2{,}006$ to produce $Q = 24$ and $D = 304$.

5.6 Summary and Conclusion

This chapter describes our enhanced simulated annealing technique for the single-row routing problem. The problem has its origin in the complex VLSI design. Massive connections between pins and vias in the circuitry require their breakdown into smaller single-row components, as described by So (1974). The components are solved individually to generate solutions which can then be combined to provide the overall solution to the problem. Single-row routing is a divide-and-conquer approach that is frequently used in solving the VLSI design problem.

In this chapter, we propose ESSR (*Enhanced Simulated annealing for Single-row Routing*) for solving the single-row routing problem. The main objective in this problem is to produce a realization that minimizes both the street congestion and the number of doglegs. Simulated annealing is based on the thermostatic cooling properties of atoms in physical systems. By performing slow cooling, the nets in the single-row routing problem align themselves according to a configuration with the lowest energy. In general terms, the energy is represented as the absolute sum of the heights of the net segments. This energy is directly proportional to both the street congestion and the number of doglegs. In our approach, we pivot the street congestion value while having the energy drops directly proportional to the number of doglegs. This action has the effect of minimizing the number of doglegs as the energy drops.

Our experiments using ESSR produce optimal solutions with both minimum street congestion and minimum number of doglegs in most data sets, especially in cases where the number of nets is large. The results match well against our previous method (Salleh and Zomaya, 1999), and the methods by Tarng *et al*. (1984), and Du and Liu (1987).

CHAPTER 6

SINGLE-ROW TRANSFORMATION OF COMPLETE GRAPHS

6.1 Introduction

Complete graph is a fully-connected graph where every node is adjacent to all other nodes in the graph. Very often, many applications in science and engineering are reducible to this type of graph. Hence, a simplified form of a complete graph contributes in providing the solutions to these problems. In this chapter, we present a technique for transforming a complete graph into a single-row routing problem. Single-row routing is a classical technique in VLSI design that is known to be NP-complete. We solved this problem earlier using a method called ESSR, as described in the previous chapter. The same technique is applied to the present work to transform a complete graph into a single-row routing representation. We also discuss the application of this work on the channel assignment problem in the wireless cellular telephone networks.

In many cases, problems in engineering and other technical problems can be reduced as problems in graph theory. A problem of this nature is said to be reducible to the form of vertices and links of a graph, and the solution to the problem can be obtained by solving the graph

problem. Furthermore, several solutions to the problems in graph theory have found their roots in some well-known prototype problems, such as the the travelling salesman problem, the shortest path problem and the minimum spanning tree problem. Solutions to these problems are provided in the form of dynamic programming techniques, mathematical programming and heuristics. Most of these prototype problems have been proven to be NP-complete and, therefore, no absolute solutions to the problems are established. However, their reduction to the form of graphs have, in some ways, simplified their complexity and pave way for further improvement to their solutions.

In this chapter, we study the relationship between a complete graph and its single-row representation. A complete graph is a graph where every vertex in the graph is adjacent to all other vertices. As described in the previous chapter, single-row routing is a classical problem about finding the most optimum routing from a set of *terminals*, or nodes, arranged in a single-row. In the *Very Large Scale Integration* (VLSI) technology, the terminals are the pins and vias, and the routes consist of non-intersecting horizontal and vertical tracks called *nets*. The main goal in single-row routing is to find a realization that reduces the congestion in the network.

We also propose a model for transforming a complete graph as nets in a single-row axis. The motivation for this proposal comes from the fact that some problems in engineering are reducible to the form of a complete graph. We study the mapping properties of a complete graph into its single-axis representation, in the form of the single-row routing problem. We devise a strategy for mapping this graph, and then apply the method for solving a graph-reducible problem, namely, the channel assignment problem in the wireless cellular telephone networks.

Our chapter is organized into five sections. Section 6.1 is the introduction. Section 6.2 describes the problem in the chapter, while in Section 6.3, we outline the details of the mapping strategy for converting the complete graph into its single-row axis representation. Finally, Section 6.4 discusses the application of this method on the channel assignment problem. We conclude the chapter with the summary and conclusion in Section 6.5.

6.2 Problem Formulation

This chapter discusses the tranformation of a complete graph as a single-row routing problem. It is easy to verify that a complete graph, C_m, with m vertices has $m(m-1)$ links (or edges). This is because each vertex in the graph has a degree of $(m-1)$. The problem begins with the mapping of the links in this graph as terminals in a single-row axis. Single-row routing problem is an important component in finding an optimum routing in VLSI design (Raghavan and Sahni, 1983). The single-row representation, S_m, of the graph, C_m, consists of m zones and $m(m-1)$ terminals, all aligned in a single-row axis. The terminals are to be formed in equally-spaced intervals along the single-row axis. In VLSI, each terminal represents a pin or via. In the single-row routing problem, nets joining pairs of terminals are to be formed to allow communication between the terminals. A net is made up of non-intersecting horizontal and vertical lines that is drawn in the order from left to right.

In order to produce a practical area-compact design, the nets have to be drawn according to the routes that will minimize the wiring requirements of the network. The main objective in the single-row routing problem is to determine the most optimum routing between pairs of the terminals so as to reduce the congestion in the whole network. It is also important that the routing is made in such a way that the interstreet crossings (doglegs) between the upper and lower sections of the single-row axis be minimized. This is necessary as the terminals in the single-row axis are very close to each other, and a high number of interstreet crossings will generate an intolerable level of heat that may cause problems to the network. In optimization, the problem of minimizing the congestion in the network reduces to a search for the right orderings of the nets, based on a suitable energy function.

6.3 Complete Graph Partitioning Strategy

A graph G consists of a set of vertices and edges, connecting some of these vertices. A graph where a path exists to connect any two pairs of vertices in the graph is called a *connected graph*, otherwise it is a *disconnected graph*. Node j in the graph having d links with its neighbors is said to have a degree of d_j. A graph with m nodes where every node is a neighbor of every other nodes in the graph is a complete graph, C_m. In C_m, every vertex has the same degree of $m-1$.

6.3.1 Formation of Zones and Terminals from a Complete Graph

In C_m, every link between a pair of vertices in the graph is mapped as a terminal in S_m. Therefore, a C_m graph having m vertices and $m(m-1)$ links is mapped into m zones with a total of $m(m-1)$ terminals in S_m. A vertex with degree j in the graph occupies a zone in S_m with d_j terminals.

We outline the overall strategy for mapping a complete graph. In general, the transformation of a complete graph, C_m, into its single row representation, S_m, consists of two main steps. First, the vertices, v_j, are mapped into the zones, z_j, that are numbered according to their vertex number, j, for $j = 1,2,...,m$. The next task is to determine the number of terminals in each zone, z_j, in S_m, which is simply the degree, d_j, of its corresponding vertex, v_j, in C_m. Finally, we obtain the complete layout of S_m by combining all the terminals from each zone and number them successively beginning from the first zone to the last.

Our method for creating the zones and their terminals in S_m from a complete graph, C_m, is outlined in `Algorithm 6.1`, as follows:

```
/* Algorithm 6.1: Formation of zones and terminals in S_m from C_m */
Given a complete graph C_m;
Draw the zones, z_j, in S_m, which corresponds to v_j in C_m, for j = 1,2,...,m;
for j = 1 to m
      Determine the degree, d_j, of every vertex, v_j, in C_m;
      Set i = 1;
      for k = 1 to d_j
            Set the terminal number, t_i = i;
            Update i ← i + 1;
```

6.3.2 Construction of Nets from a Complete Graph

In the previous section, we described a plan to form the zones and nets in S_m from C_m using `Algorithm 6.1`. We illustrate the idea on the problem of forming a single-row representation of C_5, a complete graph with $m = 5$ vertices. In this problem, each vertex in the graph has a degree of 4. There are $m = 5$ zones, z_i, for $i = 1,2,...,5$ and the number of terminals on the single-row axis is $m(m-1) = 20$. Hence, the number of nets formed is $r_m = \frac{m(m-1)}{2} = 10$.

Figure 6.1 shows the zones and terminals in S_5 formed from C_5 when `Algorithm 6.1` is applied.

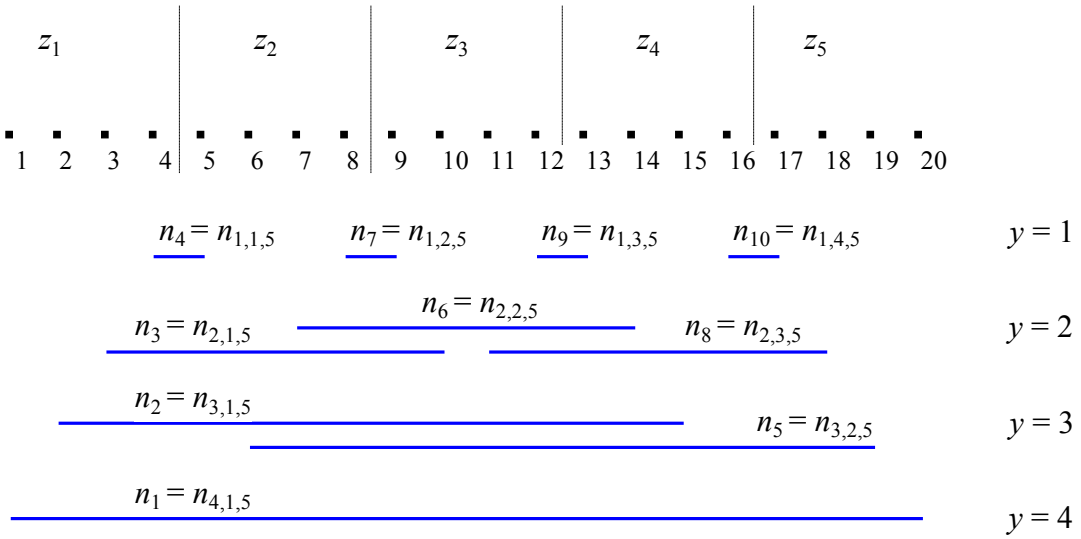


Fig. 6.1: Formation of nets based on the zones and levels in S_5 from C_5

We now present a technique for forming the nets in the network that will contribute in minimizing the total energy in S_m. The technique calls for the formation of the nets by grouping them first into several levels based on their width. The *width* of net k, denoted as w_k, is defined as the difference between its beginning and end terminals, given as $w_k = e_k - b_k$. A *level,* y, in S_m consists of a set of equal-width nets grouped in ascending order from the lowest width to the highest. Our strategy begins with Proposition 1 which consists of first forming levels where the nets with equal width are grouped. In this proposal, the nets in S_m are created by defining their end-points. Once the nets have been formed, the next step consists of sorting and renumbering the nets based on their beginning terminals, in ascending order from the lowest to highest. These two steps are to be described later in `Algorithm 6.2`.

Preposition 1: The i th net in level y in S_m, denoted as $n_{y,i,m} = (b_{y,i,m}, e_{y,i,m})$, formed from the complete graph, C_m, is grouped into levels based on its width, $w_{y,m}$, according to the following relationships:

$$b_{y,i,m} = (m-y) + (m-1)(i-1), \tag{6.1a}$$

$$e_{y,i,m} = b_{y,i} + w_{y,m} \tag{6.1b}$$

for $y = 1,2,...,m-1$, and $i = 1,2,...,m-1$.

From Proposition 1, we obtain the width of the nets in level y of S_m, given as follows:

$$w_{y,m} = 1 + (m+1)(y-1), \tag{6.2}$$

and the number of nets in each level as follows:

$$r_{y,m} = (m-y). \tag{6.3}$$

The strategy for grouping the nets into levels based on their width is to minimize the total network energy, given earlier in Equation (5.6). This goal can be achieved by forming nets starting from the shortest width, continue with the next shortest and so on. Starting with level 1, that is, $y = 1$, the nets are formed from two consecutive terminals from two different zones. This level has the most minimum width possible, given by $w_{1,m} = 1$. This minimum width has the advantage of producing the net energy equals 0, as the net can be drawn directly on the node axis. The i th net is formed from the last terminal in z_i and the first terminal in zone $(i+1)$ th, to

make sure that the width remains the same. Using Equations (6.1a) and (6.1b) from Proposition 1, we then obtain the i th net in this level, $n_{1,i,m} = (b_{1,i,m}, e_{1,i,m})$, given as $b_{1,i,m} = (m-1) + (m-1)(i-1)$ and $e_{1,i,m} = b_{1,i} + 1$.

In level 2, the first net is obtained by having the second last terminal in z_1 as its beginning terminal, and the second terminal of z_2 as the ending terminal. This gives the width as $w_{2,m} = 1 + (m+1) = m+2$. In general, the i th terminal in this level, $n_{2,i,m} = (b_{2,i,m}, e_{2,i,m})$, is given by $b_{2,i,m} = (m-2) + (m-1)(i-1)$ and $e_{2,i,m} = b_{2,i} + w_{2,m}$.

/* **Algorithm 6.2:** Construction of nets in S_m */
Given a complete graph C_m with m vertices;
Let the number of nets in level 1, $r_1 = r$;
The initial width of nets in level 1 is 1, that is, $w_{1,m} = 1$;
for $y = 1$ **to** r
  if $y > 1$
    Update the width of the nets in level y, $w_{y,m} \leftarrow 1 + (m+1)(y-1)$;
    Update the number of nets in level y, $r_{y,m} \leftarrow (m-y)$;
  for $i = 1$ to r_y
    Form the ith net in level y as follows:
      Update the left terminal of net y, $b_{y,i,m} \leftarrow (m-y) + (m-1)(i-1)$;
      Update the right terminal of net y, $e_{y,i,m} \leftarrow b_{y,i} + w_{y,m}$;
for $y = 1,2,...,r$
  for $i = 1,2,...,r_y$
    Sort $(b_{s,i,m}, e_{s,i,m})$ in ascending order with $b_{s,i,m}$ as the primary key;
for $k = 1$ to $\frac{m(m-1)}{2}$
  Assign $n_k = (b_k, e_k)$ from the sorted $(b_{s,i,m}, e_{s,i,m})$;

Algorithm 6.2 above summarizes our method for constructing the nets based on the levels of the nets with equal width. In this algorithm, the nets are formed based on Equations

(6.1a) and (6.1b). The number of nets and their width in each level are determined from Equations (6.2) and (6.3), respectively. Once the nets have been formed, the algorithm then sorts and renumbers the nets based on their beginning terminals, in ascending order from the lowest to highest. `Algorithm 6.2` prepares the nets before the next important step, which is their execution in ESSR to determine their optimum routing.

We illustrate the idea of constructing the nets using Proposition 1 through an example with a complete graph of five vertices, C_5. The zones and terminals are obtained by applying `Algorithm 6.1`. By applying Equations (6.1a) and (6.1b) from Proposition 1, we obtain the nets grouped into 4 levels, as shown in Figure 6.1. `Algorithm 6.2` transforms the C_5 into S_5, and the results are shown in Table 6.1. We then apply `ESSR` to the nets to obtain the results in the form of an ordering with minimum energy, $E = 11$, as shown in Figure 6.2. The final realization of the network with $Q = 3$ and $D = 1$ is shown in Figure 6.3.

Table 6.1: Formation of nets in S_5 from C_5

Level, y	Width, $w_{y,5}$	#nets, r_y	nets
1	1	4	(4,5), (8,9), (12,13), (16,17)
2	7	3	(3,10), (7,14), (11,18)
3	13	2	(2,15), (6,19)
4	19	1	(1,20)

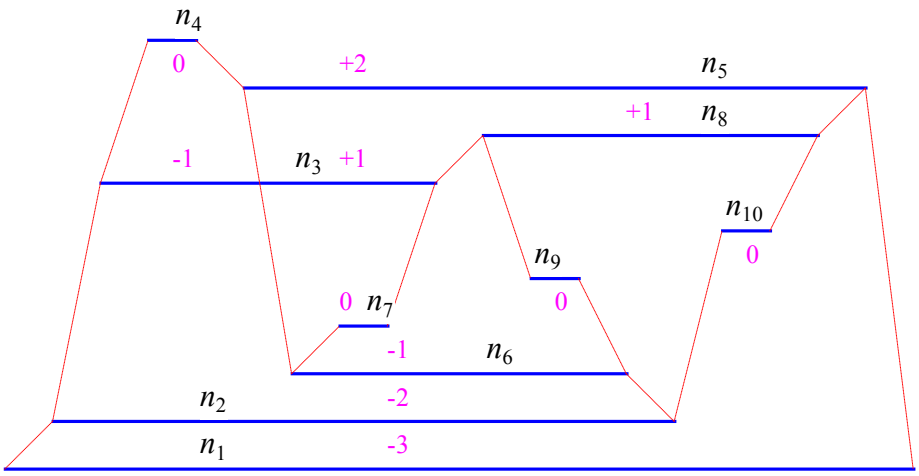


Fig. 6.2: Nets ordering with minimum energy, $E = 11$, of C_5 using Algorithm ESSR

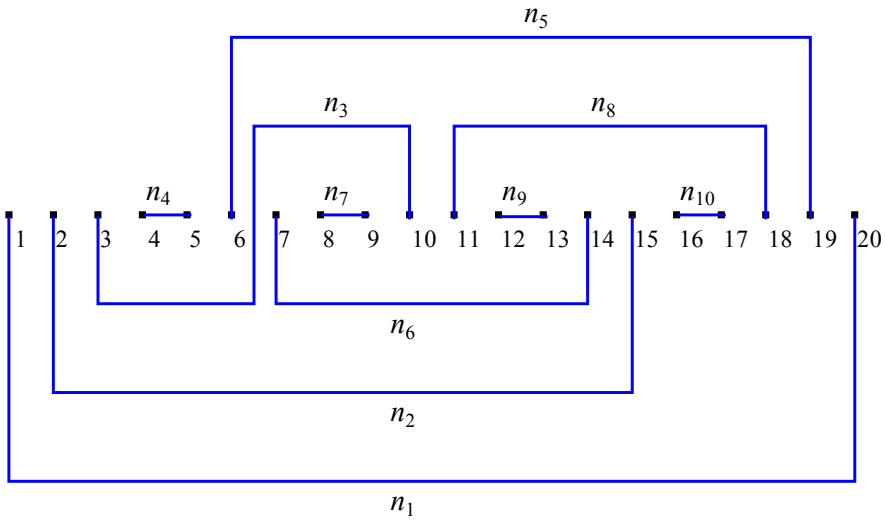


Fig. 6.3: Final realization of C_5 with $E = 11$, $Q = 3$ and $D = 1$

We also apply the method to several other models of complete graphs. Table 6.2 summarizes the results of these graphs with m vertices, C_m, in their single-row representations. Figure 6.4 shows the final realization of the routing obtained using ESSR from C_{10}.

Table 6.2: Summary of results for some complete graphs, C_m

C_m	#nets	E	Q	D
$m = 5$	10	11	3	1
$m = 6$	15	28	4	40
$m = 8$	28	128	9	21
$m = 10$	45	403	16	53

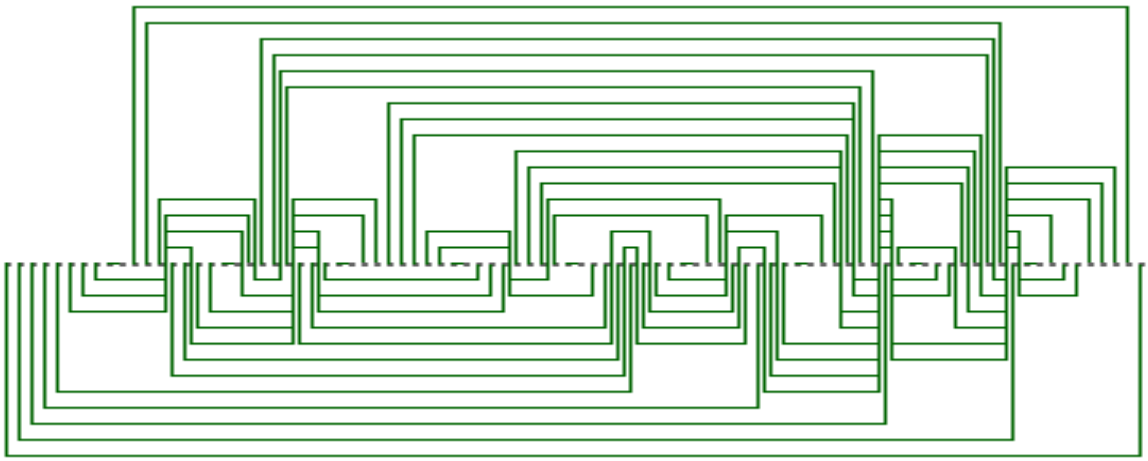


Fig. 6.4: Realization of an optimum assignment of 45 nets from C_{10} from Table 6.2.

6.4 Application on the Frequency Assignment Problem

In this section, the single-row mapping strategy is applied to the problem of assigning radio channels in a wireless cellular telephone network. In the wireless cellular telephone network (Mathar and Mattfeldt, 1993), the assignment of radio frequencies to the mobile users within the network can be modeled as the problem of mapping a complete graph into non-intersecting single-row nets. This network consists of a geographical region partitioned into several cells where each cell is allocated with a *base station* (BS). A base station has a transmitter and a receiver for receiving and transmitting messages from/to mobile users within its cell. The base stations in the network are linked with high-speed cables to the *mobile switching center* (MSC), which functions as a controller to allow communication between any two mobile users in the network by assigning a channel each to each of them. When a call is made from a cell, a request is received by the base station of the cell. The base station relays this request to the mobile switching center (MSC). Assuming the call is made and received within the network, a channel each needs to be assigned to the caller and the receiver. In this network, MSC plays an important role in assigning a pair of different channels to both the caller and the receiver, to allow immediate circuit switching.

In this problem, we model the channels as the edges of a complete graph. The cells in the network are then represented as nodes in the graph. In the single-row axis, each of these cells is a zone and the channels allocated to a cell are terminals in the zone. Communication between two mobile users from two different cells is established through a net linking their two terminals. We model the single-row communication to be handled by the mobile switching center. This is because MSC has a control on all channel assignments in the network, and this important task must be done immediately without delay when requests for calls are received. In addition, MSC must also be able to provide services associated with problems in channel assignments, such as location finding of mobile users, and channel handovers as a mobile user moves from one cell to another.

We illustrate our model using an example with a network of five cells. The problem reduces to the complete graph, C_5, which is represented as the zones, z_j, for $j = 1,2,...,5$ in S_5, as shown in Figure 6.1. Hence, 20 channels are available for assignments and each of these channels is represented as a terminal in the single-row axis. The channels are formed using the same technique discussed in Section 6.3, to produce the results as shown in Table 6.1 (the channels are numbered by assuming there are no electromagnetic interferences on the channels) . Figure 6.4 is then the final realization of the optimum routing of the nets obtained using ESSR.

6.5 Summary and Conclusion

In this chapter, we propose a method for transforming a complete graph, C_m, into its single-row representation, S_m. We first describe the single-row routing problem, which is a classical technique in VLSI design. We relate the problem to our earlier work which solved the problem using a method based on simulated annealing, called ESSR. The transformation from C_m to S_m involves the formation of nets based on Proposition 1. The proposition groups the nets with equal width which contributes in reducing the overall energy of the network. The whole process is implemented using `Algorithms 6.1` and `6.2`. We then apply ESSR to the network to obtain some reasonably good results for optimum routing. Finally, we also describe briefly the application of this transformation technique in solving the channel assignment problem in the wireless cellular telephone networks.

APPENDIX

The enclosed CD-ROM has three main programs:

1. DSRM: this is our simulation program described in Chapter 3.

   Name of program: **DSRM.exe**

   Description: This model schedules randomly generated tasks dynamically on the 4×4 reconfigurable mesh network. Tasks arrive at every time slot with randomly determined length, and are mapped to the available processors according to the DSRM parallel algorithm.

2. Our simulation program described in Chapter 4.

   Name of program: **IPRM.exe**

   Description: This model performs edge detection on a given image using the 4×4 reconfigurable mesh network using the Laplacian convolution technique.

3. Our simulation program described in Chapter 5.

   Name of program: **SRR2.exe**

   Description: This program is the simulation model of ESSR on a network of 80 nets (160 pins). At every temperature mark, 10 nets are chosen at random and are swapped according to the simulated annealing process.

REFERENCES

Aarts, E. and Korst, J. (1989); "Simulated annealing and Boltzmann machines", *John Wiley*, New York.

Chow, Y. and Kohler, W.H. (1979); "Models for dynamic load balancing in heterogeneous multiple processing element systems", *IEEE Trans. Computers*, vol.28, no.5, pp.354-361.

Deogun, J.S. and Sherwani, N.A. (1988); "A decomposition scheme for single-row routing problems", *Technical Report Series #68*, Dept of Computer Science, Univ. of Nebraska.

Du, D.H. and Liu, L.H., (1987); "Heuristic algorithms for single row routing", *IEEE Trans. Computers*, vol.36 no.3, pp.312-319.

Efford, N. (2000); "Digital image processing:, Addison-Wesley.

El-Rewini, H.; Lewis, T.G. and Ali, H.H. (1994); "Task scheduling in parallel and distributed systems", *Prentice Hall*.

Hwang, J.; Chow, Y.; Anger, F.D. and Lee, C. (1989); "Scheduling precedence graphs in systems with interprocessor communication times", *SIAM J. Computing*, vol.18, no.2. pp. 244-257.

Kuh, E.S., Kashiwabara, T. and Fujisawa, T. (1979); "On optimum single-row routing", *IEEE Trans. Circuits and Systems*, vol.26, no.6, pp.361-368.

Lewis, T.G. and El-Rewini, H. (1992); "Introduction to parallel computing", *Prentice-Hall*.

Flynn, M.J. (1972); "Some computer organizations and their effectiveness", *IEEE Trans. Computers*, vol.21, no.9, pp. 948-960.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, (1983); “Optimization by simulated annealing”, *Science*, vol.220, no.4598, pp.671-678.

Kobayashi, H. (1978); “Modeling and analysis: an introduction to system evaluation methodology”, Addison-Wesley.

Kolmogorov, A.N.K. (1957); “On the representation of continuous functions of many variables by superimposition of continuous functions of one variable and addition”, *Dokl. Akad. Nauk SSSR*, vol.114, pp.953-956.

Lin, H. and Raghavendran, C.S. (1991); “A dynamic load-balancing policy with a central task dispatcher”, *IEEE Trans. Software Engineering*, vol.18, no.2, pp.148-158.

Mathar, R. and Mattfeldt, J. (1993); “Channel assignment in cellular radio networks”, *IEEE Trans. Vehicular Technology*, vol.42 no.4.

Miller, R. and Prasanna-Kumar, V.K. (1993); “Parallel computations on reconfigurable meshes”, *IEEE Trans. Computers*, vol.42, no.6, pp.678-692.

Nakano, K. and Olariu, S. (1998); “An efficient algorithm for row minima computations on basic reconfigurable meshes”, *IEEE Trans. Parallel and Distributed Systems*, vol.9, no.8.

Olariu, S., Schwing, J.L. and Zhang, J. (1993); “Fast component labelling and convex hull computation on reconfigurable meshes”, *Image and Vision Computing*, vol.11, no.7, pp.447-455.

Olariu, S., Schwing, J.L. and Zhang, J. (1995); “A fast adaptive convex hull algorithm on two-dimensional processing element arrays with a reconfigurable bus system”, *Computational Systems Science and Engineering*, vol.3, pp.131-137.

Olariu, S. and Zomaya, A.Y. (1996); “A time- and cost-optimal algorithm for interlocking sets with applications”, *IEEE Trans. Parallel and Distributed Systems*, vol.7, no.10, pp.1009-1025.

Raghavan, R. and Sahni, S. (1983); “Single row routing”, *IEEE Trans. Computers*, vol.32, no.3, pp.209-220.

Ramamritham, K. and Stankovic, J.A. (1989); “Distributed scheduling of tasks with deadlines and resource requirements”, *IEEE Trans. Computers*, vol.38, no.8, pp. 1110-1122.

Rutenbar, R.A. (1989); “Simulated annealing algorithms: an overview”, *IEEE Circuits and Devices Magazine*, January, pp. 19-26.

Saletore, V. (1990); “A distributed and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks”, *Proc. of DMCC-5*, Portland, Oregon, pp.994-999.

Salleh, S., Sanugi, B., Jamaluddin, H., Olariu, S. and Zomaya, A.Y. (2002); "Enhanced simulated annealing technique for the single-row routing problem", *Journal of Supercomputing*, vol. 21 no.3, pp.285-302.

Salleh, S. and Zomaya, A.Y. (1999); “Scheduling for Parallel Computing Systems: Fuzzy and Annealing Techniques”, *Kluwer Academic Publishers*, Boston.

So, H.C. (1974); “Some theoretical results on the outing of multilayer printed wiring board”, *Proc. IEEE Symposium on Circuits Systems*, pp. 296-303.

Stout, Q.F.; “Ultrafast parallel algorithms and reconfigurable meshes”, Proc. DARPA Software Technology Conference, 1992, pp.184-188.

Tarng, T.T., Sadowska, M.M. and Kuh, E.S. (1984); “An efficient single-row algorithm”, *IEEE Trans. Computer-Aided Design*, vol.3 no.3, pp.178-183.

Ting, B.S., Kuh, E.S. and Shirakawa, L. (1976); "The multilayer routing problem: algorithms and necessary and sufficient conditions for the single row, single layer case", *IEEE Trans. Circuits Systems*, vol. 23, pp. 768-778.

LIST OF PUBLICATIONS

1. **Shaharuddin Salleh**, Mohd Ismail Abdul Aziz and Stephan Olariu, “Single-row transformation of complete graphs”, submitted to the *Journal of Supercomputing,* USA, 2002.

2. **Shaharuddin Salleh**, Ruzana Ishak and Nurul Huda Mohamed, “Single-row routing technique for channel assignments in wireless cellular networks”, *Proc. Simposium Kebangsaan Sains Matematik* 10, ISBN #983-52-0284-2, Johor Bahru, 23-24 Sept. 2002, pp. 310-315.

3. **Shaharuddin Salleh**, Bahrom Sanugi, Hishamuddin Jamaluddin, Stephan Olariu and Albert Y. Zomaya, "Enhanced simulated annealing technique for the single-row routing problem", *Journal of Supercomputing*, Kluwer Academic Publishing, vol. 21 no.3, pp.285-302, March 2002.

4. **Shaharuddin Salleh**, Nur Arina Bazilah Aziz, Nor Afzalina Azmee and Nurul Huda Mohamed**,** “Dynamic multiprocessor scheduling for the reconfigurable mesh networks”, *Jurnal Teknologi*, vol.37(C) , December 2002, pp.57-68.

5. **Shaharuddin Salleh**, Nur Arina Bazilah Aziz, Nor Afzalina Azmee and Nurul Huda Mohamed**,** “Dynamic multiprocessor scheduling for the reconfigurable mesh networks”, *Proc. Malaysian Science and Technology Congress*, November 8-10 2001, Penang.

6. **Shaharuddin Salleh**, Nor Afzalina Azmee, Nurul Huda Mohamed and Nur Arina Bazilah Aziz, “Reconfigurable meshes model for the edge detection problem”, *Proc. Malaysian Science and Technology Congress*, November 8-10 2001, Penang.