Automated Wire Routing Using Parallel Processing

Hesham Mohamed Abdel Moneim Keshk

Kyoto University, Japan

1996

Abstract

Automated wire routing is one step of computer aided design (CAD) for either integrated circuit (IC) or printed circuit board (PCB). The main goal is to find a path connecting between terminals of every network (net) and this path must not intersect with any other path. The main problems of automated wire routing are the long computation time and large memory size required.

Even though sequential computers are getting faster and with larger memory, the chips that the designers are attempting to design are growing in a much faster rate. Parallel computers can offer a solution for this problem. Unless good and efficient parallel algorithms for CAD applications are developed, the parallel computers will not be able to deliver their computational power to VLSI and PCB designers.

This thesis introduces a parallel wire routing algorithm for message passing parallel computers. The whole routing area is divided among the PEs (Processor Elements) to decrease the required memory for each PE. The efficiency of parallelism depends on how to divide the whole routing area into partitions, and how to assign these partitions to the PEs. It is required to reduce the communication cost between processors by decreasing number of messages and increasing the overlapping between communication and computation.

As short length nets take short time for routing, the communication overhead in routing short nets should be much worse than long nets even if only one message is used per net. To overcome this problem, we introduce a new router which consists of two phases with two different methods of layout partitioning and assigning to the processors. The first phase is aimed to route all short nets using few messages, then

the rest of the nets will be routed in the second phase.

In the first phase, The routing area is divided into square partitions. One partition is assigned to each PE. Each processor routes all the nets which lie inside its own partition without any communications with the other PEs. We rotate the assigned area to each processor to route all short nets which has a length shorter than a certain value. Only few messages are used during this phase.

In the second phase, a slice based partitions assignment is introduced to route long nets. We divide the whole routing area into layers, divide each layer into slices, and assign one or more slice to each PE. Dividing the routing area into layers increases the parallelism between nets and decreases the number of vias used (higher routing quality). Dividing each layer into slices reduces the communication cost for routing long nets. The number of messages per net in this phase is proportional with number of vias.

With the increasing complexity of the routing problems, large problems are decomposed into two processes: global and detailed. In the global routing, the whole routing space is divided into partitions, and it is required to determine in which partitions each net will be routed. The exact path for each net within the partitions is found in the detailed routing. As many previous introduced routers separate completely global and detailed routing, the global router has to estimate the number of available tracks (capacity) inside each partition. The accuracy of this estimation has a great effect on the successful connection ratio.

This thesis introduces a new cooperative relation between global and detailed routers to improve the successful connection ratio. The detailed router calculates the exact capacities of all partitions, and feeds these capacities back to the global router. This cooperative relation allows the global router to get the actual number of available tracks inside the partitions instead of estimating these capacities.

For detailed routing, the well known maze running algorithm has a serious disadvantage that previous routed net paths may prevent other nets from being routed later. A new detailed routing algorithm called max-capacity algorithm is also

introduced to improve the successful connection ratio. Max-capacity algorithm finds the maximum capacities for the slices on the global path, and routes each net within these slices by a path which decreases each capacity by only one.

Several programs are implemented on a message passing parallel computer and evaluated using MCM benchmarks. The results show that using two phase algorithm decreases the communication cost. Max-capacity algorithm obtains higher successful connection ratio than maze running algorithm. The successful connection ratio has been further improved by using the cooperative relation between global and detailed routers. A connection ratio of 100% has been obtained using our algorithm.

Contents

List of Figures

List of Tables

Acknowledgment

"AL-HAMD-LEL-LAH"; Thanks to the God who guided me to his way, and gave me the help to do this work.

I would like to express my sincere gratitude and indebtedness to Professor Shinji Tomita who accepted me as a doctor candidate in his laboratory in Kyoto University, and provided me the opportunity to compelete this research. I am realy grateful and thankful to him for his careful guidance, encouragement, valuable discutions and critcism.

I deeply appreciate the interesting comments and useful discussions from Associate Professors Hiroshi Nakashima and Shin-ichiro Mori of Kyoto University. Thanks are also due to all the members of Professor Tomita's laboratory for their discussions and supports throughout this research.

I gratefully acknowledge the Ministry of Education of Japan for awarding me the scholarship. I really acknowledge the Egyptian goverment for the additional financial support. I would like to thank Fujitsu Laboratories Ltd for offering me the chance to use their parallel computer to implement my parallel algorithms.

Last, but not the least, I am very grateful to my parents and my wife for caring and encouraging me these many years. I am forever greatly indebted to them for all what they did for me.

Chapter 1

Introduction

1.1 Parallel algorithms for CAD

The study of parallel algorithms for CAD applications is important for two reasons:

1. In view of the increasing complexity of VLSI circuits, there is a growing need for sophisticated CAD tools to automate the different steps in the design of the VLSI or PCB systems. Parallel processing offers an effective solution to handle the large complexity of VLSI and PCB designs of the future.

2. CAD applications represent a rich class of important, non-numerical, unstructured, applications that are generally difficult to parallelize effectively. Researchers of parallel compilers are typically exposed to parallel algorithms for structured numerical problems, such as matrix multiplication, and often fail to solve a large non-numerical, unstructured problems in parallel [1].

1.2 Overview of VLSI CAD

The VLSI design process can be viewed as a sequence of transformations on behavioral, structural, and physical design representations. Behavioral representations describe outputs of circuits as functions of their inputs. Structural representations describe the composition of circuits in terms of components and interconnections among them. Physical representations are characterized by information used in the manufacture and fabrication of physical systems, such as geometric layout or topological constraints.

A top-down design methodology of electronic systems divides the design process into phases. The phases are functional design, logic design, circuit design, and physical design. Figure 1.1 shows an example of the representation used during each phases of VLSI design [1].

Automated layout (physical design) of wiring segments for printed circuit boards (PCBs) and integrated circuits (ICs) consists of two main functions: *Placement*: arranging the components (cells or modules) on the plane, and *Routing*: connecting these components with wiring.

Although placement and routing are intimately related and interdependent, they have been traditionally separately solved because of their computational complexities. Hence placement algorithms use some crude measures to estimate the routing complexities of various placements. The routing algorithm sub-sequentially takes a given placement and performs the exact wiring of the networks (**nets**). Both placement and routing are NP complete; hence a large number of heuristics exist for solving the problems, which vary in their requirements and qualities of layout produced. Unfortunately, routing algorithms that produce high quality layouts often take long computation time. Recently, researches have therefore turned to parallel processing to solve complex routing problems quickly and with high quality [1].

1.3 Automated wire routing

Automated wire routing is to find a path connecting between terminals of each network and this path must not intersect with any other path on the same layer. The locations of these terminals have been determined during the preceding placement process. Figure 1.2a shows an example of one layer grid with four nets. The paths connected these net terminals could be as shown in Figure 1.2b.

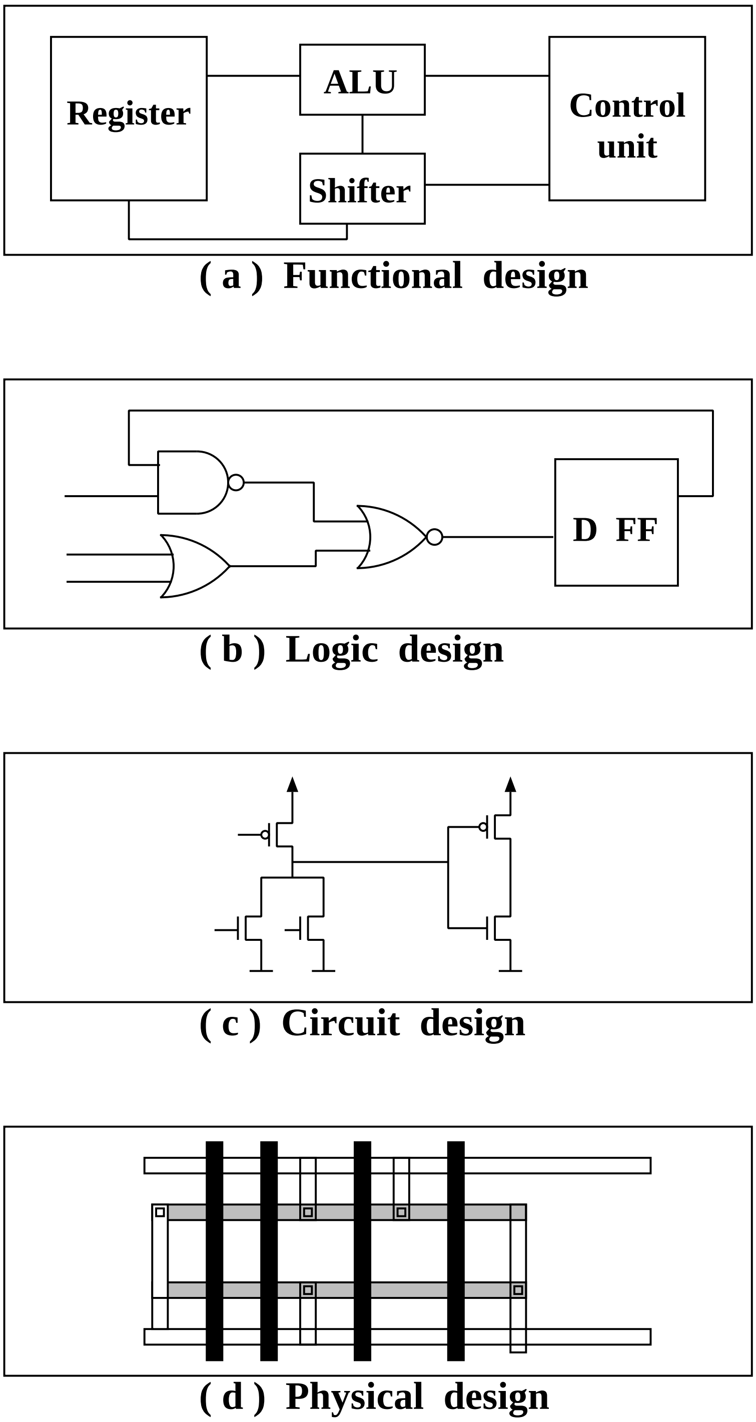


Figure 1.1: Phases of VLSI design.

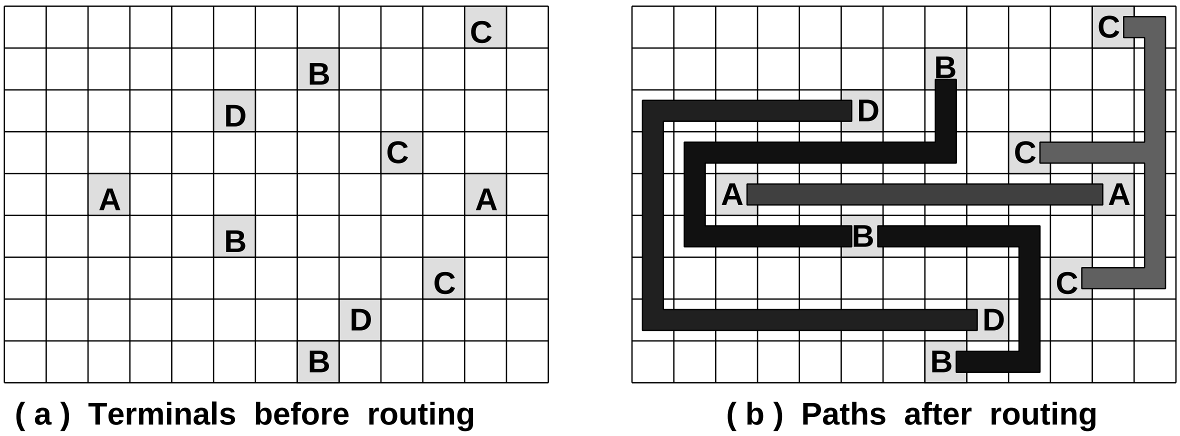


Figure 1.2: Example of one layer routing problem with four nets.

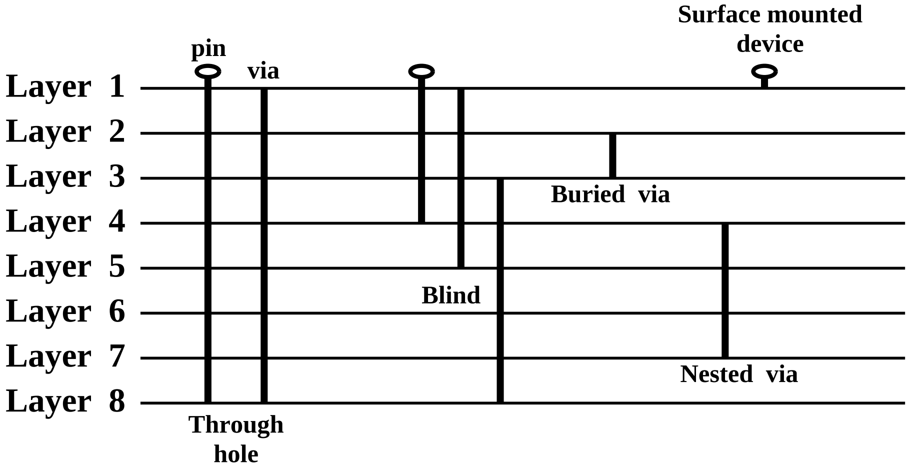


Figure 1.3: Different kinds of vias and pins.

For a printed circuit board (PCB), the components to be routed are integrated circuit (IC) packages, and the electrical connections are made with a metal etching process. Metal is uniformly deposited on a carrier (typically fiberglass) surface and the unnecessary metal etched away, leaving wire lines. Connection between layers are made by drilling holes through the fiberglass and plating them with metal. Recently, surface mounted devices become widely used and a variety of via structures (Figure 1.3) becomes possible in addition to the traditional through hole vias [2].

In an integrated circuit, wire lines of polysilicon are fabricated to carry electrical signals between circuits. In addition, one or more layers of metal separated by insulating layers are deposited and etched above the silicon to form wire lines. Holes are left in the oxide to make connection between layers.

In both cases, the routing problem is very similar. The router must determine the wire lines patterns to be laid out and the positions of the vias to interconnect the layers [3].

In general there are many sets of points that must be electrically connected. The routing problem is to connect all of the points in each set and ensure that the wiring paths of the different sets do not intersect on any layer. In addition to these requirements, there are often several aspects to be optimized. Long wire lengths cause propagation delays, so meandering paths should be avoided. Sometimes, there are critical signals whose delay must be minimized at the expense of others. Available routing space is often variable, in which case overall area should be minimized. The particular feature to be optimized depends on the layout problem to be solved [3].

1.4 Maze running algorithm

The most famous routing technique which is able to handle all the varied requirements for routing in a flexible manner is the maze running algorithm which is firstly proposed by Lee [4]. Although Lee's algorithm was the first algorithm to solve the automated wire routing problem (1961), it is still used until now. Maze running algorithm is a breadth first search algorithm, and it guarantees to find the shortest path between two points if one exists. It can easily be modified to find a path which satisfies some other requirements.

The basic problems of maze running algorithm are the long computation time and large memory size required. For a grid plane with dimensions $K \times K$, maze running algorithm requires $O(K^2)$ memory, and it also requires $O(K^2)$ running time in the worst case. Some researches [5] [6] improve the speed by using hardware implementation of maze running algorithm.

1.5 Parallel processing

Recently, several researches have tried to overcome the long time and large memory space problems by using parallel processing [7] [8] [9]. Parallelism could be exploited in both nets and space (grid area). In the former, nets are assigned to the PEs (processor elements), so that each PE will be responsible for routing its own nets [10]. In the later, the whole routing area is divided among the PEs so that several PEs are sharing in routing one net [11] [12] [13]. To increase the parallelism, these two kinds of parallelism are exploited together [7] [14].

For message passing computers, the whole routing area is usually divided among the PEs to decrease the required memory for each PE. The efficiency of parallelism depends on how to divide the whole routing area into partitions, and how to assign these partitions to the PEs. It is required to reduce the communication cost between processors by decreasing number of messages and increasing the overlapping between communication and computation. Different methods for partitioning the routing area have been introduced, but there is a trade-off between processor utilization and inter processor communication [12].

1.6 Global & Detailed routing.

With the increasing complexity of the routing problems dealing with thousands of nets, the large problems should be decomposed into two processes[1]: global and detailed. In the global routing, the whole routing area is divided into partitions, and it is required to determine in which partitions each net will be routed. The exact path for each net within the partitions is found in the detailed routing.

As many routers [13] [15] separate completely global and detailed routing, the global router has to estimate the number of available tracks (capacity) inside each partition. It is very hard to get an accurate estimation of these tracks, though the accuracy of this estimation has a great effect on the successful routing ratio.

1.7 Outlines of this thesis

This thesis introduces a parallel global and detailed wire router for message passing parallel computers.

In Chapter 2, we give an introduction to parallel computer architectures and algorithms. We explain different classifications of parallel architectures on the basis of: 1: how the machine relates its instructions to the data being processed, 2: how the memory are connected to the processors, and 3: how the processors are connected together. We also describe some fundamentals of parallel algorithms like the different kinds of parallelism and the load balancing problem. An introduction to AP1000 parallel computer is also included.

In Chapter 3, we give a background of sequential routing algorithms for both detailed and global wire routing. We explain different routing algorithms like maze running algorithm and line search algorithm. We discus also several modifications of maze running algorithm which have been proposed to overcome the problems of the long execution time and large memory size needed by the original algorithm.

In Chapter 4, we introduce two new algorithms to parallelize maze running algorithm.

These algorithms have high degree of parallelism inside and obtain high speedup specially if the Manhattan distances between the terminals of the nets are short as compared with the whole routing area (layout). They are suitable for routing small size routing areas. Each of them divides the routing area among the processors by a new technique which is suitable for mesh network computers. In the first algorithm, moving boundaries are used in dividing the routing area. The second algorithm consists of two phases. In the first phase, we rotate the areas assigned to the processors to route all short nets. In the second phase we use the competing processors algorithm [10] to route the remaining nets. We implemented our two algorithms and two other algorithms which have been introduced by Takahashi et al [10] [11] on the AP1000 parallel computer. The results and comparisons between the four algorithms are also included.

Chapter 5 introduces a new parallel wire routing algorithm called Amon which can be used for large scale multi-layer layout. Amon consists of two processes: global and detailed. Both of these processes are done simultaneously in two stages pipeline. The PEs are divided into global PEs and detailed PEs. We use maze running algorithm and its modifications for both global and detailed routing. In the global routing, the nets are divided among the PEs to route them in parallel. In the detailed routing, to overcome the problem of the large memory size required by maze running algorithm to represent the routing area, the whole routing area is divided among the detailed PEs. The multi-layer grid is divided into layers, each layer is further divided into slices, and then one or more slices are assigned to each detailed PE. The detailed PEs route nets, which has a non overlapping global paths, in parallel within their own slices. Dividing grid into layers increases the parallelization between nets and decreases the number of vias used (higher routing quality). Dividing each layer into slices decreases the communication cost between PEs by using fewer messages.

In Chapter 6, we introduce two new modifications of Amon to improve the successful routing ratio of Amon. By using these modifications, the routing ratio becomes almost the same like the best sequential routing algorithm. Unfortunately these modifications slightly decrease the parallelism and increase the computation time. The first modification introduces new cooperative relation between global and detailed routers. This cooperative relation allows the global PEs to know the actual numbers of available tracks (*capacities*) inside the partitions instead of estimating these capacities. The second modification is a new algorithm for detailed routing called maximum capacity algorithm (*max-cap*). Max-cap algorithm finds the maximum capacities for the slices, which are on the global path of the net, and routes each net within these slices by a path which decreases each capacity by only one. Using Max-cap algorithm for detailed routing obtains higher connection ratio than using maze running algorithm.

In addition to improving the routing ratio, we try to improve the speed and to decrease the communication cost by introducing two phases algorithm (Amon2). In the first phase, we use the rotating area algorithm introduced in Chapter 4 to route short nets using only few messages. In the second phase, we use the slice based grid partition algorithm introduced in Chapter 5 to route the rest of the nets. An improvement in the speed is obtained by this merge without affecting the routing ratio.

In Chapter 7, the conclusion for this thesis and suggestions to future work are stated.

Chapter 2

Parallel architectures and algorithms

In this chapter, we give an introduction to parallel computer architectures and algorithms. We explain different classifications of parallel architectures on the basis of: 1: how the machine relates its instructions to the data being processed, 2: how the memory are connected to the processors, and 3: how the processors are connected together. We also describe some fundamentals of parallel algorithms like the different kinds of parallelism and the load balancing problem. An introduction to AP1000 parallel computer is also included.

2.1 Introduction

Parallel processing is a fast growing technology that permeates many areas of computer science and engineering. It is natural that the concurrencies which abound in many application domains start to reflect in computer systems. We live in a concurrent and complex universe, and as we master computer technology better, we tend to bring this concurrency to our computers. Supercomputers as well as smaller computers rely more and more on parallel processing methods. Parallelism not only brings unprecedented speed and cost effectiveness, but also raises a new set of complex and challenging problems to solve. Most of these problems are software related [16].

A parallel computer is built by connecting several low cost processors and memories together through an interconnection network. The basic idea of a parallel computer is that a number of processors work in cooperation on a single task. The motivation is that if one processor takes an amount of time T to do a task, then P processors can do the same task in time T/P. On real problems, this perfect speedup could not be achieved because of communication, synchronization and load unbalance.

Parallel computing is becoming an increasingly cost effective and affordable means for providing enormous computing power and represents a challenge to costly conventional supercomputers. A parallel computer is termed a massively parallel processor **MPP** if it has 1000 or more processors. A large number of medium priced moderately parallel (using 10 to 100 processors) multiprocessors are also commercially available today. Numerous vendors of workstations are currently marketing multiprocessor based workstations.

Although it is possible to build parallel machines whose peak performance will be thousands of MFLOPS (millions of floating point operations per second) and MIPS (millions of instructions per second), it is extremely difficult to harness the computational power effectively to obtain all those MFLOPS and MIPS to the user. It is required to design good and efficient parallel algorithms that can effectively use the hardware resources to get the maximum performance [1].

2.2 Classification of parallel computer architecture

There is a large variety of general purpose parallel processor architectures which can be characterized by different parameters.

2.2.1 Flynn's classification

There are a number of alternative ways in which parallel computers can be classified. Michael J. Flynn (1966) introduced a scheme for classifying the architecture of a computer on the basis of how the machine relates its instructions to the data being processed. Flynn's classifications are described below with some explanations [17]:

1. **SISD:** single instruction stream, single data stream.

   SISD computers are conventional serial computers in which the instructions are executed one by one and a single instruction deals with at most one data operation.

2. **SIMD:** single instruction stream, multiple data stream.

   In SIMD computers, a single instruction may initiate a large number of operations. These instructions are executed once at a time but are able to work on several data stream at once. A central controller broadcasts the same instruction to all processors. Each processor then executes the instruction on its data.

3. **MISD:** multiple instruction stream, single data stream.

   The MISD class implies several instructions operating simultaneously on a single data item. It is generally agreed that there is no computer that at the present can be classified as MISD. MISD is sometimes used to describe pipeline processors.

4. **MIMD:** multiple instruction stream, multiple data stream.

   MIMD computers are characterized by the simultaneous execution of more than one instruction on several data stream. The individual processors run under the control of their own programs, which allows great flexibility in the tasks the processors are doing at any given time.

Note that synchronization of the individual processors in SIMD systems is carried out by the controller, while in MIMD systems other mechanisms must be used to ensure that the processors are doing their tasks in a correct order with the correct data.

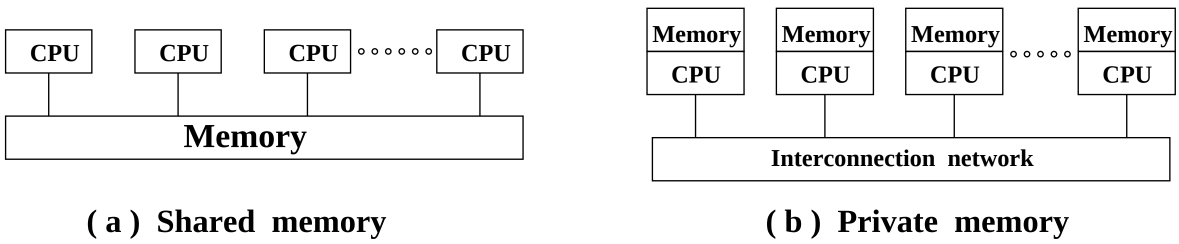


Figure 2.1: Logical memory organization.

2.2.2 Shared memory versus private memory

Multiprocessor systems can be further classified into shared memory multiprocessors and private memory multicomputers as shown in Figure 2.1.

In a shared memory (tightly coupled) system, all processors share a common logical address space and they can access all memory locations. Each processor is connected to the common memory via an interconnection network. All communication between individual processors is through the common memory. A major advantage of shared memory system is potentially very rapid communication of data between processors. A serious disadvantage is that different processors may wish to use the common memory simultaneously, in which case there will be a delay until the memory is free. This delay, called contention time, is increased as the number of processors increases.

In a private memory (loosely coupled) system, each processor has its own local memory, and each processor can address only its own local memory. The processors communicate with each other via **message passing** system, in which data is transfered between processors. Any synchronization and data access of nonlocal data have to be performed using explicit message passing between processors.

For message passing systems, consider the problem of sending a data from the memory of one processor (P_1) to the memory of another processor (P_2). First, this data is loaded into a buffer memory or collected into contiguous locations in memory of P_1. Then, a send command will be executed on P_1 to transfer the data to a buffer or memory of P_2. P_2 will execute a receive command and the data will be stored to

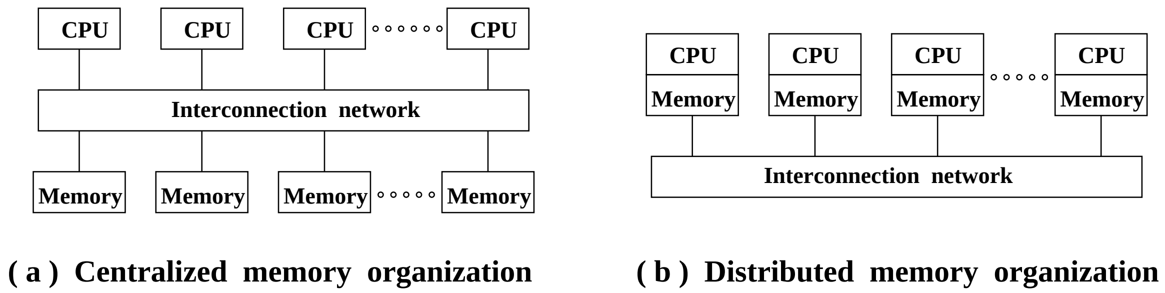


Figure 2.2: Physical memory organization.

their final location in the memory of P_2. Co-processors may be available to relieve the main processors from much of this work.

2.2.3 Centralized memory versus distributed memory

Even though shared memory multiprocessors provide a logical view of a uniform shared single address space, the processors and memories are physically implemented in one of two ways; centralized or distributed. The two organization are shown in Figure 2.2.

In the centralized memory organization, the memory is centralized in a pool. The processors are on one side of the interconnection network, and the memory module are on the other side.

In the physical distributed memory organization, the processors and memories are physically located together at one node, and multiple nodes are connected together by interconnection network.

2.2.4 Interconnection schemes

The topology of interconnection network can be either static or dynamic. Static networks are formed of point to point direct connections which will not change during program execution. Dynamic networks are implemented with switched channels, which are dynamically configured to match the communication demand in user programs.

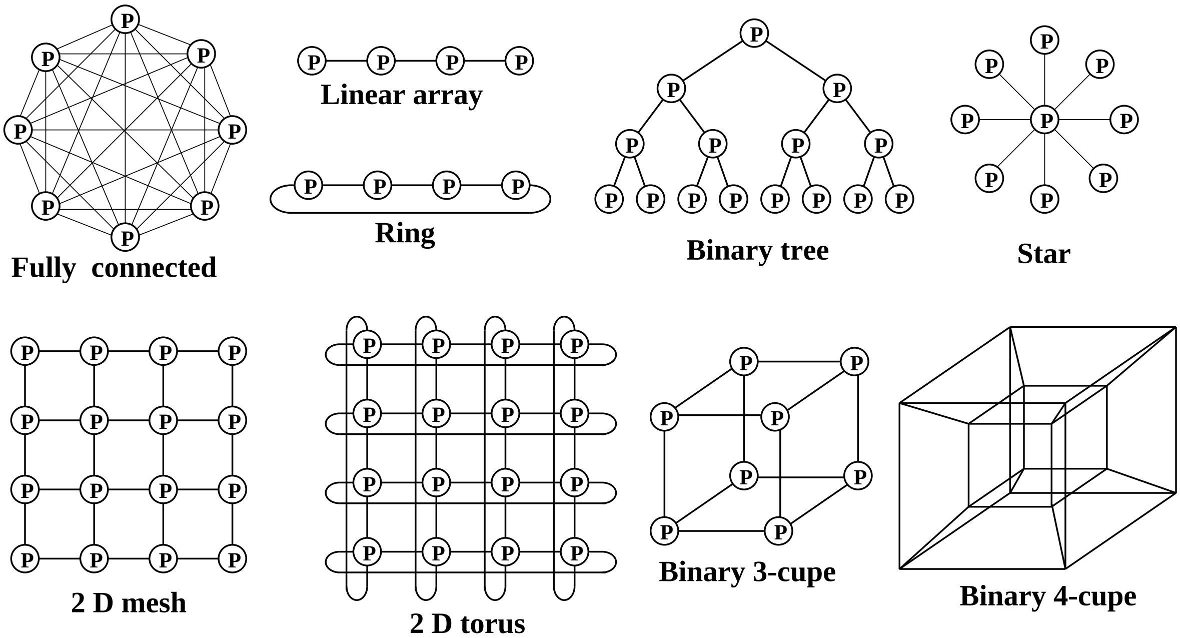


Figure 2.3: Static connection networks.

Static connection networks

Static networks use direct links which are fixed once built. The most common interconnection schemes are shown in Figure 2.3 [18] [19]:

Fully connected: Each processor has a direct connection to every other processor. A completely connected system of p processors requires $p-1$ lines emanating from each processor, which is impractical if p is large. The number of edges incident on a node is called the *node degree.*

linear array: This is a one dimensional network in which N nodes are connected by $N-1$ links in a line. Each processor is connected to two nearest neighbors (except the end processors, which are connected to only one). An advantage of this connection pattern is its simplicity, with at most two lines emanating from each processor. A severe disadvantage is that data may need to be passed through several processors to reach its final destination. The maximum number of transmission that must be made to communicate between any two processors of the system is called the communication length or diameter of the system. The diameter for N nodes linear array is $N-1$ which is rather long for large N.

Ring: A ring is obtained by connecting the two terminal nodes of a linear array with one extra link. The communication length is $N/2$ for a bidirectional ring. Adding more links increases the node degree and decreases the network diameter.

Binary tree: A binary tree of 15 nodes in four levels is shown in Figure 2.3. In general, a K level binary tree should have $N = 2^{K-1}$ nodes. The maximum node degree is 3 and the diameter is $2(K-1)$.

Star: One node is connected to all the other nodes. The star has a high node degree $(N-1)$ and a small constant diameter.

Mesh: Each processor is connected to only a few neighboring processors. Most mesh-connected arrays used a two dimensional connection pattern, in which the processors are laid out in a regular two dimensional grid and each processor is connected to its north, south, east, and west neighbors. The communication length for N processors arranged in a square array is $2(\sqrt{N}-1)$. This communication length can be decreased by adding more connections, called local links. Three dimensional array is an example where each processor is connected to its six nearest neighbors. The addition of more local links leads to faster average communication but at the expense of additional complexity.

Torus: In torus network, the edge processors of a mesh connection are connected together in a two dimensional ring. A two dimensional torus with N nodes has a node degree of 4 and a diameter of $2(\sqrt{N}/2)$.

Hypercube: Consider the connection pattern in three dimensions illustrated in Figure 2.3. Here, the processors are vertices of a cube in 3D space and the edges of the cube are the local links between processors. Each processor is connected to its three nearest neighbors in the sense of nearest vertices on the cube.

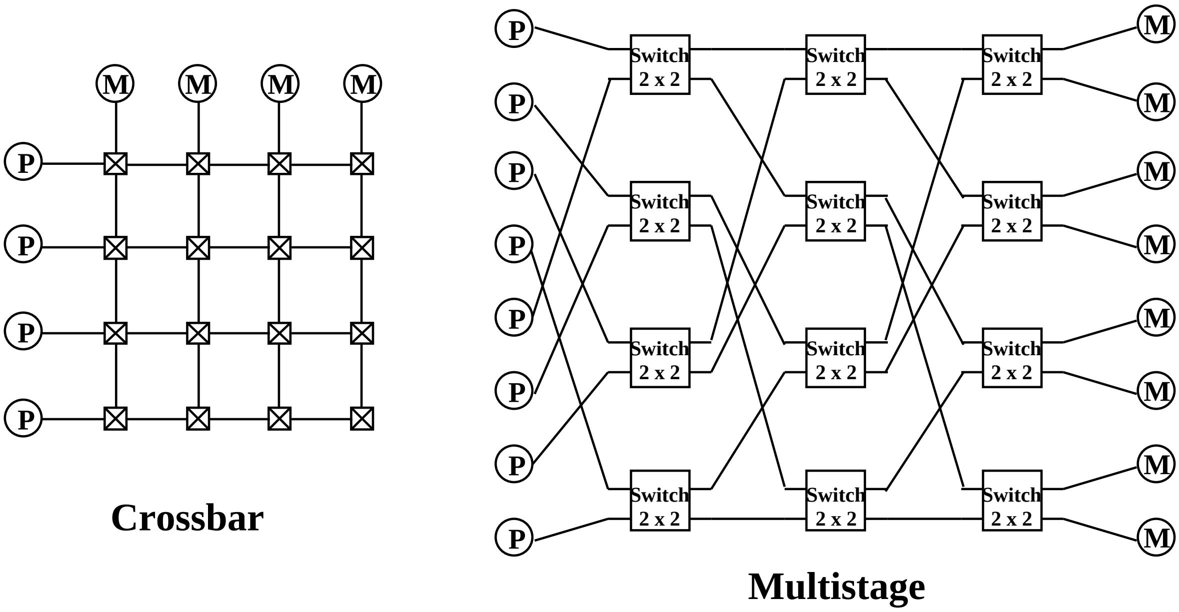


Figure 2.4: Dynamic connection networks.

Now imagine the analogous connection scheme using a cube in k dimensions. Again, the processors would be visualized as the 2^k vertices of this k dimensional cube. Each processor is connected to its k adjacent vertices (Hamming distances equal 1), along the edges of the cube. Such a connection pattern is called a hypercube or binary k cube connection. A k cube can be constructed by connected all corresponding processors of two $(k-1)$ cubes.

For N processors hypercube, both the number of connections per processor and the communication length are equals to $\log_2 N$. As the size of cube is increased, the complexity and the number of lines emanating from each processor also increase and a practical limit to the cube size will eventually be reached.

Dynamic connection networks

In dynamic connection networks, instead of using fixed connection, switches are used along the connecting paths to provide dynamic connectivity.

Crossbar switch: Each processor can be connected to each memory through switches that make the connection (Figure 2.4. This has the advantage of allowing any processor access to any memory with a small number of connection lines. But the number of switches to connect p processors to p memories is p^2 which become impractical for large p.

Multistage network: A rather general way of connecting processors, or connecting processors to memories, is by means of two ways switching network. Figure 2.4 represents an example of switching network (omega) implementation of a shared memory system, in which each processor can access each memory. Alternatively, the memories could be the processors themselves. In this case, the figure would depict a message passing local memory system.

Using a two ways switches, we need $p/2$ switches at each switching stage and $\log_2 p$ stages. This is very favorable as compared with the p^2 switches needed for the crossbar switch.

2.3 Parallel algorithm basics

Given a problem, a serial algorithm provides a precise step by step description of the computational steps that need to be performed sequentially to solve it. A parallel algorithm provides a precise description of how the given problem is to be decomposed into smaller subproblems, how each of these subproblems is to be solved, how these subproblems interact among each other, and how these subproblem solutions can be merged back to solve the original problem. The purpose of parallel algorithm design is to solve the given problem faster than the original sequential algorithm. The ratio of the execution times of the sequential algorithm and the parallel algorithm is referred to as speedup of the parallel algorithm. Let us denote:

T = time for the best serial algorithm.

T_p= time for a parallel algorithm using p processors.

Speedup $S_p = T/T_p$

Efficiency or utilization $E_p = S_p/p$

If the parallel algorithm for a problem has an efficiency of 100%, then we observe linear speedups, where $S_p = p$. However in the real world, it is very hard to achieve the efficiency of 100% due to two main reasons: 1:cost of synchronization, scheduling, or communication across parallel processors; 2: suboptimal load balance among parallel processors [1] [20].

2.3.1 Data versus functional parallelism

There are two common forms of exploiting parallelism in an application, using different forms of partitioning the problem [1]. Data parallelism involves creating multiple, identical processes and assigning a portion of the data to each process. Data parallelism is appropriate for applications that perform the same operation repeatedly on large collection of data.

Functional parallelism (also called task parallelism or control parallelism) involves creating multiple processes and having them perform different operation on a shared data set.

2.3.2 Grain size

The other important classification of parallel computation is the grain size. The term grain refers to the number of instructions executed in a processor before synchronizing or communicating with another processor. Fine grain parallel processing involves synchronizing the processors after every few instructions. Coarse grain parallel processing involves synchronizing the processors after tens of thousands of instructions. More recently, the term medium grain parallel processing has been used to characterize a grain size somewhere in the middle, for example, several hundred instructions between synchronizations [1].

2.3.3 Data dependence

For a given program, we must analyze its data dependence to determine which loops can be parallelized. Consider the following loop of a C program [1]:

for ($i = 0$; $i \leq 99$; $i{+}{+}$)

$a[i] = b[i] + c[i]$;

Since each iteration of this loop performs computations of different elements of a, b, and c, each iteration can be executed in parallel. This is the basis of data parallelism. However, not all loops can be parallelized because of data dependence. For example, consider the following loop:

for ($i = 0$; $i \leq 99$; $i{+}{+}$)

$a[i] = a[i-1] + b[i]$;

This loop can not be executed in parallel since there is data dependence between different iterations. The value of $a[1]$ produced in the first iteration will be used in the second iteration to calculate $a[2]$.

2.3.4 Load balancing and scheduling

The key to efficient parallel algorithm design is to devise a method to distribute the workload as uniformly as possible among various processors. The problem of assigning specific tasks to processors is called task scheduling. The parallel tasks can be scheduled among processes using three types of algorithms:(1)prescheduling, (2)static scheduling, (3)dynamic scheduling [1].

In prescheduling, the task division is explicitly determined by the programmer before the program is compiled. Such a method cannot be varied for different sizes of data and number of processors, hence this approach is good for function partitioning, that is assigning specific tasks to specific processors.

In static scheduling, the tasks are scheduled by the processes at runtime, but are divided in some predetermined way. For a loop having N iterations using P processors (the number being determined at runtime), processor 1 will be assigned iterations from 1 to (N/P), processor 2 from $(N/P+1)$ to $(2N/P)$, and so on. Such a scheme of assigning continuous blocks of iterations is called static-blocked scheduling.

Often we have a case where successive iterations of a loop have to perform monotonically increasing or decreasing work per iteration. In that case, the previous static block scheduling will overload the processor performing the contiguous blocks of most work. An alternative method of static scheduling for better load distribution is one where we can assign iterations in a round-robin cyclic manner in a scheme called static interleaved scheduling.

In dynamic scheduling, each process schedules its task at runtime by checking a centralized task queue from which the next available tasks are removed, and new tasks are entered.

The advantage of dynamic scheduling is better load balancing, however, there is an overhead for searching through centralized task queues in the presence of contention. Static and prescheduled scheduled produces the least overhead, however, it can produce load imbalance.

Unfortunately, the two factors (synchronization/communication versus load balance) are opposing forces. Consider a scheme that runs everything on one processor even though P processors are available. Such a scheme will have poor load balance, whereas there will be no communication/synchronization cost. On the other hand, if a problem is decomposed into the finest grain of computations, such a scheme will have the best load balance, but it will also have a high cost of communication/synchronization [1].

2.4 AP1000 parallel computer

We executed our programs on AP1000 computer which is a MIMD message passing parallel computer. The system consists of 64 to 1024 processing elements and three independent communication networks. Each processor (cell) runs its own program out of its own memory and communicates with other cells by passing messages. Messages are sent from the source cell to the destination cell through the communication network.

Here, we will summarize some features of AP1000 [21].

2.4.1 AP1000 architecture

Figure 2.5 shows the AP1000 system configuration [21]. Three independent communication networks are employed for the AP1000; the torus network (T-net) for point to point communication between cells, the Broadcast network (B-net) for 1 to N communication as data distribution, and Synchronization network (S-net) for barrier synchronization.

Figure 2.6 shows the cell hardware configuration [21]. Each cell has an integer unit (IU), a floating point unit (FPU), a message controller (MSC), a routing controller RTC), a B-net interface (BIF), and 16 Mbyte of dynamic memory. Roughly 10 Mbytes are available for the user and the rest are occupied by the cell operating system. The IU and FPU are connected to a cache memory of 128 Kbyte. The DRAMC controls the 16 Mbyte of memory. The MSC, RTC, BIF, and DRAMC are connected by an internal bus called LBUS which is 32 bit.

2.4.2 Message controller (MSC)

The MSC consists of a cache controller, a fast message handler, and four DMA controllers. Line sending and buffer receiving is fast message handler which handles small messages efficiently.

2.4.2.1 Line sending

After IU makes a message body, there is a high probability that the message is in the cache memory. This message is sent directly from the cache memory to the device. Line sending transmits the data in cache memory to the RTC or BIF in the same way as flushing cache memory. If the specified data is not in cache memory, DMA is activated automatically and data is transferred from memory.

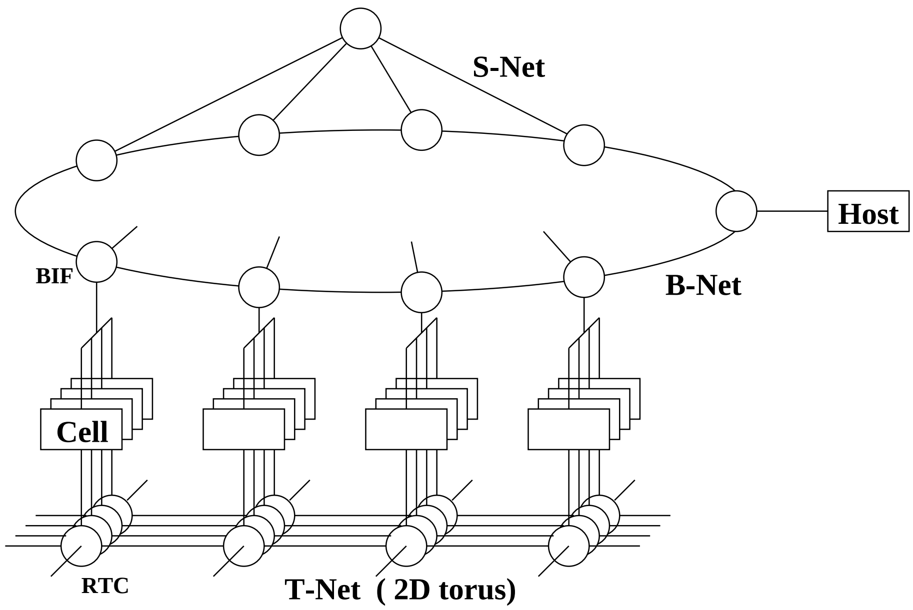


Figure 2.5: AP1000 architecture (system configuration).

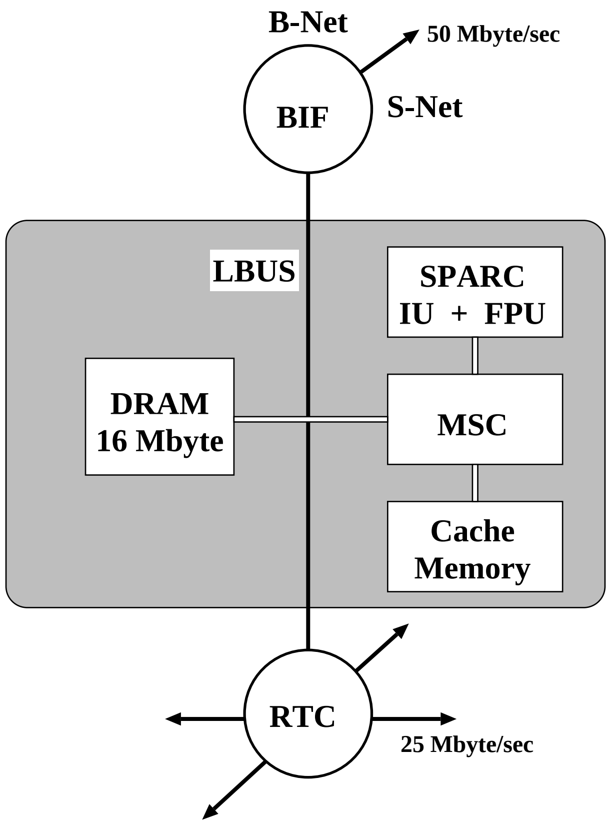


Figure 2.6: Cell configuration.

2.4.2.2 Buffer receiving

The MSC has a pair of pointer registers; one holds the read address and the other holds the write address. Using these pointers, a ring buffer in main memory can be created. The write and read pointers are managed automatically according to message receiving and message reading out by the IU. Buffer receiving is used to serve asynchronous requests for message receiving without interrupt processing by the IU.

2.4.3 Communication networks

This section is a brief introduction to the three communication networks of the AP1000 system [21].

2.4.3.1 T-net

The T-net has a two dimensional torus topology. It has a 16 bit data path and a few control signals. It has a data transfer rate of 25 Mbyte/s. Message routing is done by wormhole routing. In wormhole routing, the intermediate node stores only a few bytes called a flit. When a routing node receives the message header, the node selects the next channel of the next route. The node then transfers all subsequent flits for that message header to the route selected by the header flit.

The RTC handles routing and flow control of communications between any cells. It routes messages by fixed way; first routes a message along the X direction, then along the Y direction. The RTC can route each data item at 160 ns/word. In addition to point to point communication, the RTC has a broadcast function. A broadcast messages and point to point communication messages can be transfered concurrently.

Under the contention free condition, communication path length on the T-net has a negligible effect on communication time. Contention free means that one source cell sends a message to a destination cell while all the other cells are idle. The point to point communication time on the T-net under the contention-free condition is given by a linear function $6.9 + 6.9 \times N \mu$s, where N is the message length. Under message contention (more than one message are sent on the same channel), communication time increases as the number of messages on a communication channel increases. In the worst case, it degrades to one seventh of contention free communication time.

2.4.3.2 B-net

The B-net connects all cells and the host computer and is used for broadcasting, data distribution, and data collection. It consists of hierarchical common buses and a ring network. Up to 8 cells share the lowest level bus. Four buses are connected to the ring node which is connected to the ring network. Up to 32 cells forms a hierarchical common bus which connects with a ring node. Since the host computer is also connected by a host interface which looks like one of the ring nodes, the host computer and cells are equivalent.

The B-net consists of a 32 bit data path and some control signals and it has a data transfer rate of 50 Mbyte/s. The protocol of the B-net is similar to that of simple common bus. Before the data transfer, a cell or host trying to transfer data must assert a request to the B-net controller. And the request must wait until the request is granted.After the B-net is granted to the request, it can start data transfer.

2.4.3.3 S-net

All cells and the host are also connected by the S-net. The S-net has a tree topology and is used for barrier synchronization. The data of each cell goes up to the root of the S-net along the tree topology. At intermediate nodes, data is merged by the logical AND operation. The logical AND of the cells' data is thus calculated at the root node. The result is then distributed to all cells from the root of the S-net along the tree. The overhead for barrier synchronization is 1.6 μs to 5.2 μs.

Chapter 3

Sequential routing algorithms

In this chapter, we give a background of sequential routing algorithms for both detailed and global wire routing. We explain different routing algorithms like maze running algorithm and line search algorithm. We discus also several modifications of maze running algorithm which have been proposed to overcome the problems of the long execution time and large memory size needed by the original algorithm.

3.1 Maze running algorithm

Although Lee's algorithm [4] which is known as Maze running was the first algorithm to solve the automated wire routing problem (1961), it is still used until now because it has a high degree of flexibility. Maze running algorithm is a breadth first search algorithm, and it guarantees to find the shortest path between two points if one exists. It can easily be modified to find a path which satisfies some other requirements.

Maze is a general purpose routing algorithm that operates on a grided model and routes a single net at a time. In a grided model, the entire routing surface is represented as a uniform rectangular array of cells. The size of each cell is equal to $W + C$, where W is the wire width, and C is the clearance which must be maintained between wires. It is assumed for standardization that these two values are fixed throughout all the nets. Each cell could be used by only one net. Connecting a net is done by finding a

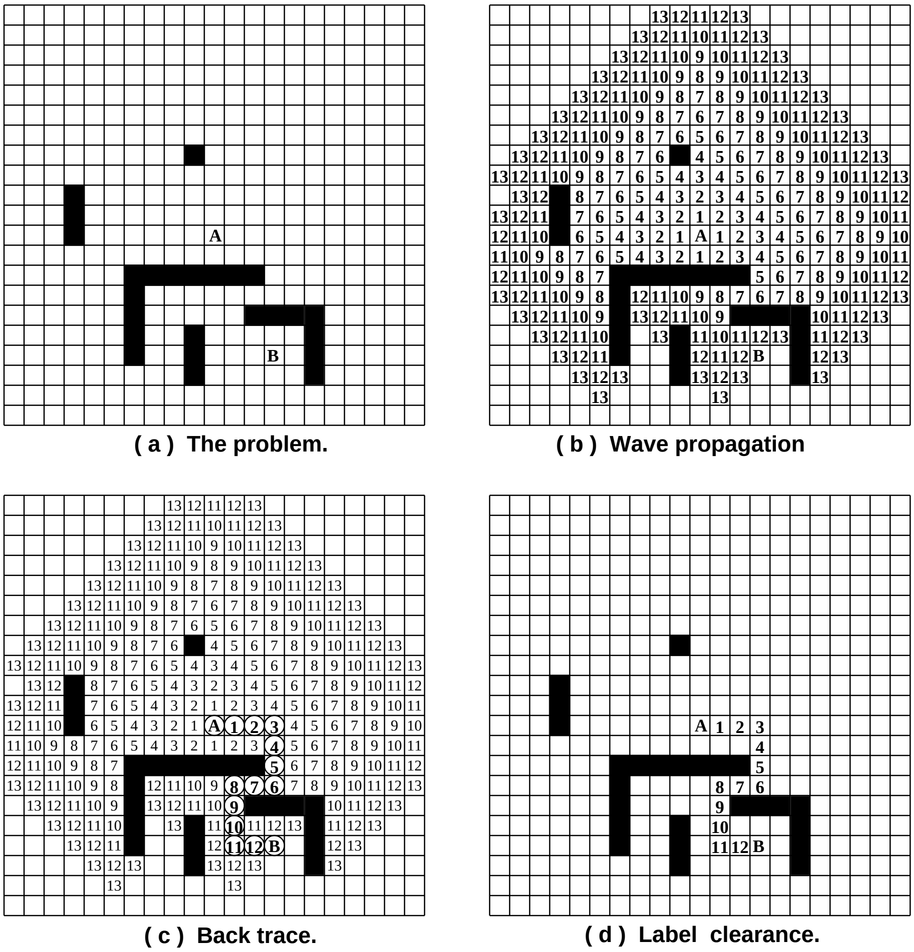


Figure 3.1: Maze running algorithm.

set of adjacent cells between the cells containing the terminals of this net [1].

Maze algorithm consists of three major phases: wave propagation (wavefront expansion), back-trace, and label clearance. For the single layer grid, the routing area are represented as a grid as shown in Figure 3.1a. To find a path between two points (A and B), we start from any point as a source (point A in this example) and search for the target (point B). The black cells are obstacles which may be some other networks which were already connected before.

In the wave propagation phase we start by entering 1 in each empty cell adjacent to point A, after that, enter 2 to each empty cell adjacent to the cells which were already labeled by 1, and so on (Figure 3.1b) until find point B or can not find any empty cells adjacent to the cells which are already labeled by m. In the back trace phase, if point B was already labeled by k, then from point B we look for a cell which is labeled by $k-1$, and from this cell look for another cell which is labeled by $k-2$, and so on until find point A (Figure 3.1c). Note that we do not change direction unless it is necessary. In the label clearance phase, all the cells except those used for the drawn path are cleared (Figure 3.1d).

3.1.1 multi-pin net

A net is called multi-pin net if it has more than two terminals. The problem to connect these terminals with a minimum total wire length is known in the graph theory as Steiner tree problem which is proved to be NP-hard [23] [24]. Maze running algorithm can be used to find sub-optimal solution [25]. As an example, to route the four terminal net shown in Figure 3.2a, one terminal is considered as a source (terminal A) while the other three terminals are considered as targets. A wave is propagated starting from point A until reached to any target (target B in Figure 3.2a). Now, a path is drawn between A and B (Figure 3.2b). All the cells on the path A-B are marked as sources and another wave is propagated starting from all these cells together (Figure 3.2c) until reached to another target (target C in this example). A new path between terminal C and path A-B is drawn (Figure 3.2d) and a new wave is propagated starting from all the cells on the path A-B-C until reached to target D (Figure 3.2e). The path A-B-C-D which connected all the four terminals could be drawn (Figure 3.2f).

The path A-B-C-D shown in Figure 3.2f is not of minimum length. This path could be improved (become shorter) by using graph theory. Removing any segment from this path divide the path into two sub-trees (Figure 3.2g). The shortest path between these sub-trees is found by applying maze running algorithm again with all the cells on one sub-tree as sources while all the cells on the other sub-tree as targets. If the shortest path is shorter than the length of deleted segment, then by inserting this path, a new

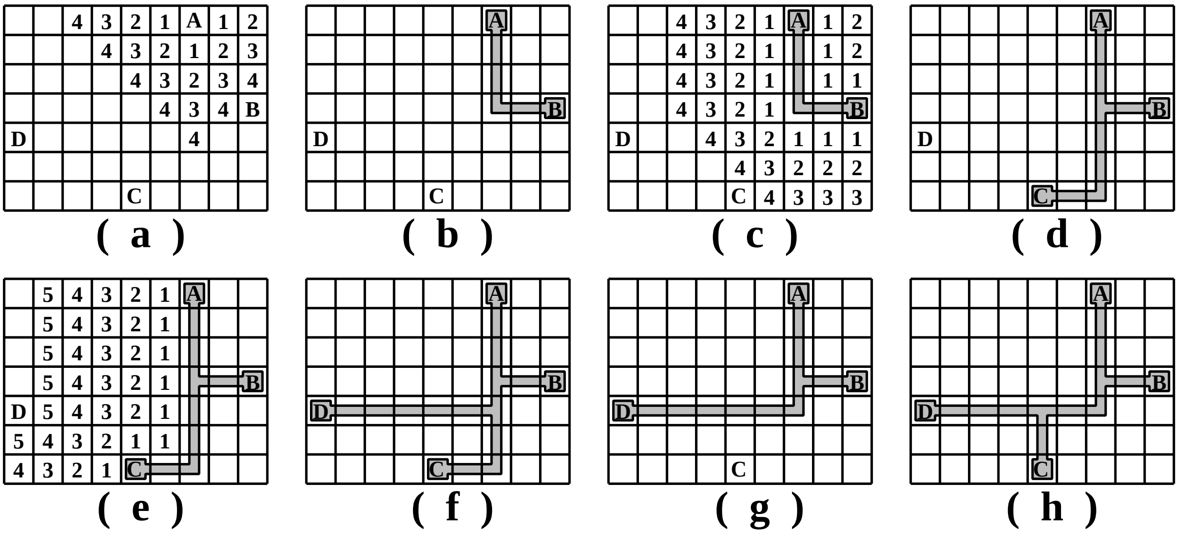


Figure 3.2: Multi-pin net.

shorter length interconnection is obtained (Figure 3.2h). This could be repeated until finding a sub-optimal shortest path connecting all the points [22].

In some cases, the previous explained method does not obtain the shortest path as shown by the example in Figure 3.3a. Watanabe et al. [26] introduce a double wave technique to find a path connected three terminals net. This technique could be iteratively applied to connect multi-terminal net which has more than three terminals.

To route a three terminal net using Watanabe double wave technique [26], one terminal (A) is considered as a source, and the other terminals (B, C) are considered as targets. A wave front with an ascending label is expanded from the source until it reaches any target (B as in Figure 3.3b). Then a wave front with a decreasing label is also expanded from this target (B) to the source (A) (Figure 3.3c). The cells which have the same label during both ascending and decreasing wave expansion is selected, these cells form a restricted area (the shaded area shown in Figure 3.3d). Another double wave expansion is made between this restricted area (as a source) and the other target (C) (Figures 3.3e,f), and another restricted area is then obtained (Figure 3.3g). A branch point P is selected by taking the logical AND between the two restricted

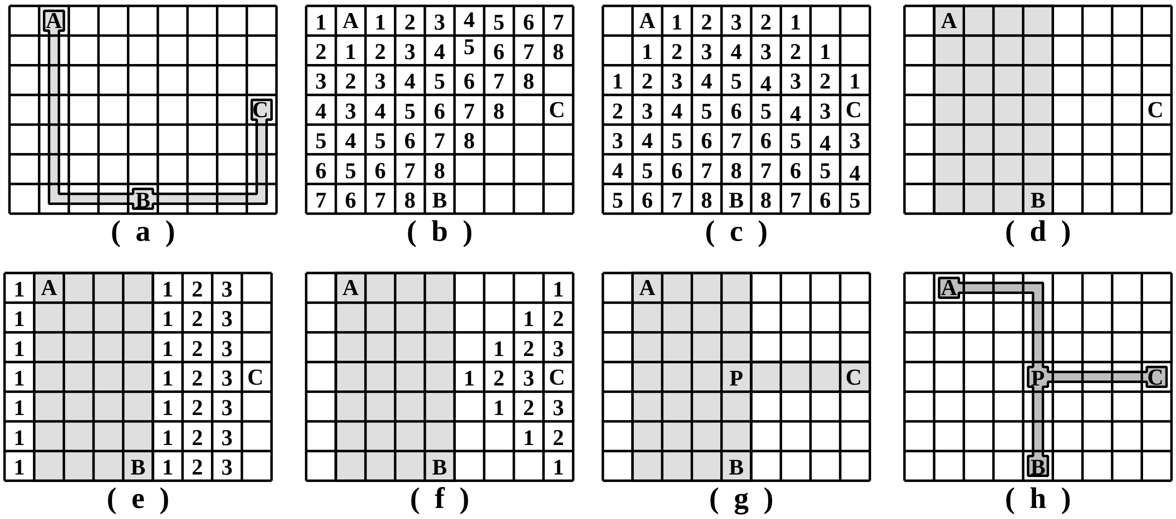


Figure 3.3: Watanabe algorithm for multi-pin net.

areas. A path connected A, B and C and passed through P is found (Figure 3.3h).

3.1.2 multi-layer

For a multi layer routing area, each cell could be considered as a unit cube as shown in Figure 3.4. For a multi-layer grid it is recommended [22] to route most of horizontal runs on some layers and most of vertical runs on the other layers. In case of two layers, routing following the horizontal-vertical rule strictly **(restricted direction)** can be achieved by blocking vertical runs on one layer and horizontal runs on the other layer at the wave propagation phase of the Lee algorithm (Figure 3.5a). A more flexible way of considering this rule is to vary the incremental weights by which the labels are increased at each step of the wave propagation, so that the desired directions are given higher priority (less weight as shown in Figure 3.5b) [22].

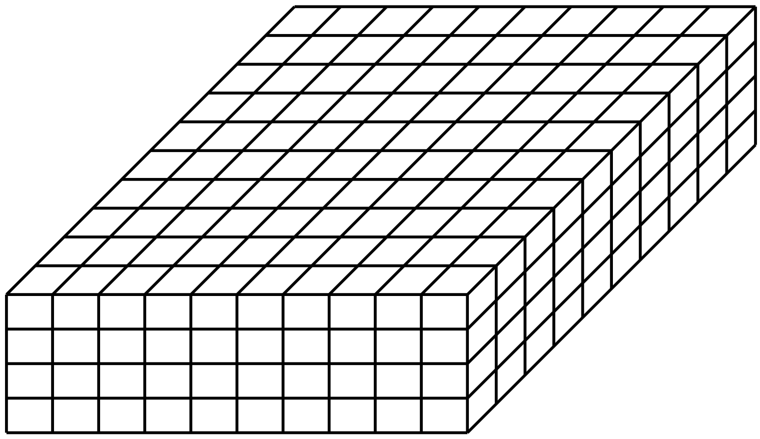


Figure 3.4: four-layers grid.

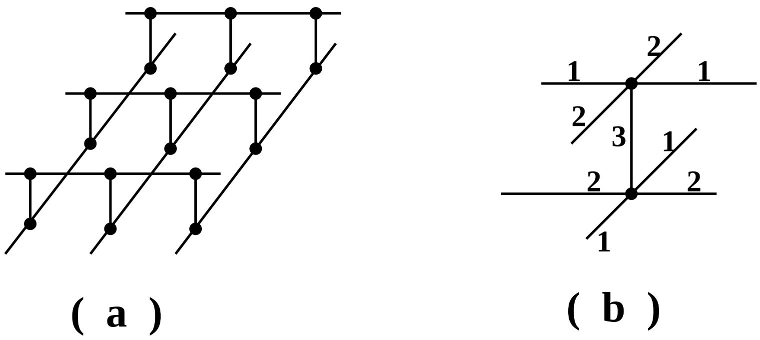


Figure 3.5: Restricted and unrestricted direction on two layers.

3.2 Line search algorithm

Another class of detailed routers called line search was proposed by Mikami and Tabuchi [27], and Hightower[28] independently. They aimed to reduce both the memory and computation time required by maze running algorithms. Line search algorithms represent the routing space and paths by a set of lines, unlike maze running algorithms in which the routing space is represented in a form of a grid and a unit of memory space is allocated for each grid point. This will reduce both the memory space and computation time than maze if the routing area is not congested. However,

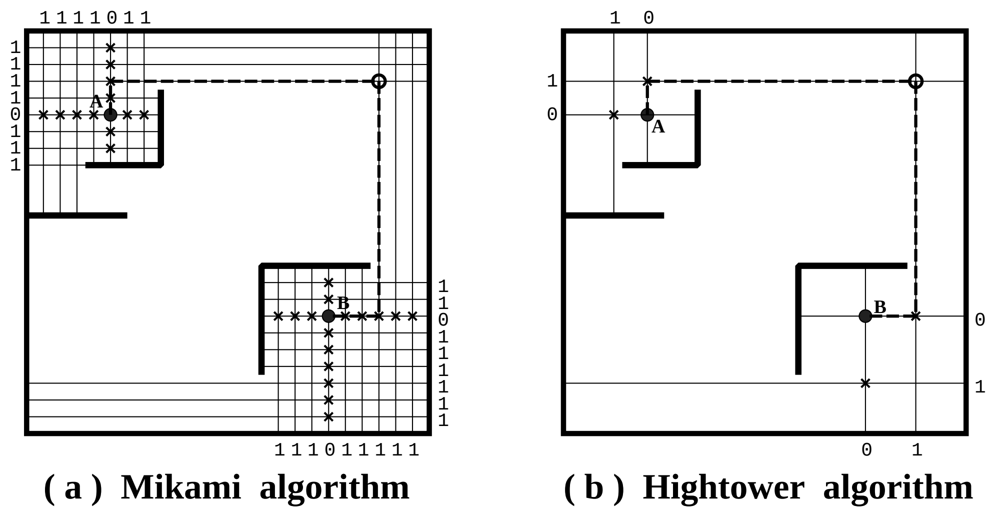


Figure 3.6: Line search algorithm.

for complicated routing problems, line search algorithm runs slower and needs more memory than maze running algorithm [22].

Let A and B be a pair of terminals of a net. The line search algorithm starts by generating lines starting from A and B in both horizontal and vertical directions. These lines are extended until they hit obstructions or the external frame of the routing space. The algorithm chooses escape points along the current lines and additional new perpendicular lines are sent out from these escape lines. This will be repeated until a line originated from A intersects another line originated from B.

The main difference between the two original line search algorithms is how to choose the escape points. In Mikami and Tabuchi algorithm, all the lines perpendicular to the preceding line are generated (Figure 3.6a). This guarantees to find a path with minimum number of bends (or vias in case of two layers) if one exists. Hightower's algorithm tries to generate a single escape point for each line by considering only the line extendable beyond the obstacle which has blocked preceding trial lines as shown in Figure 3.6b. Hightower's algorithm runs faster than Mikami's algorithm because it needs a smaller stack of escape point. On the other hand, Hightower's algorithm does

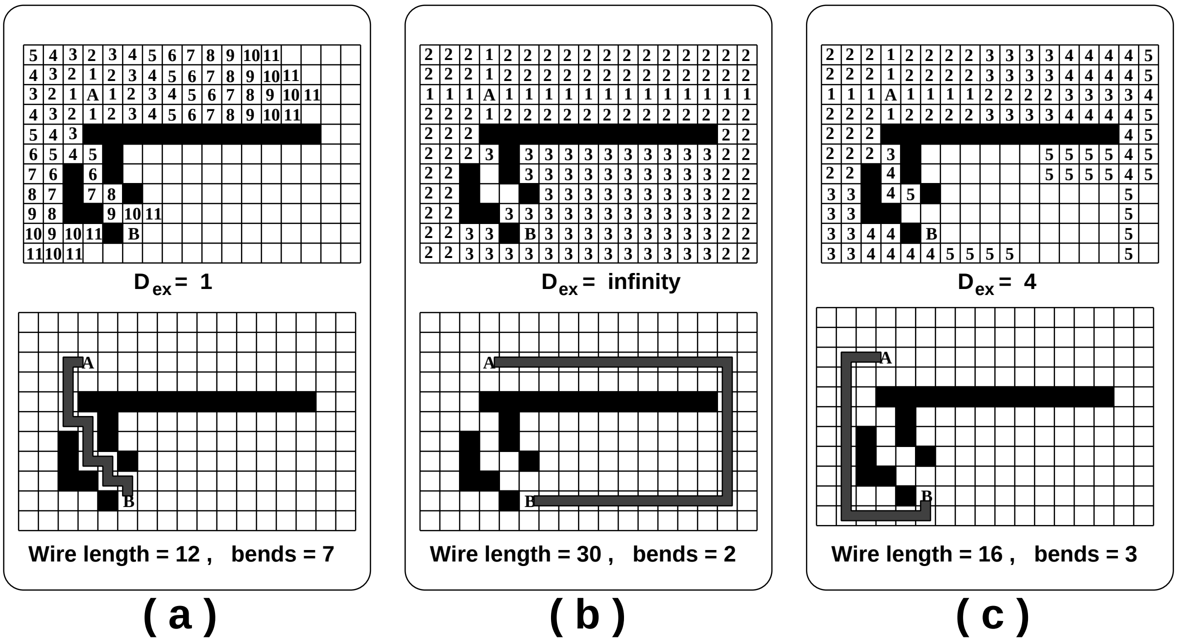


Figure 3.7: Combining maze running algorithm with line search algorithm by using expansion distance.

not guarantee to find a path even if one exist, and there is no guarantee for the paths found by this algorithm to be the paths with minimum number of bends.

3.3 Modifications of Maze running algorithm

Many researches [29] [30] modified the original maze to be faster. Some other researches [6] improve the speed by using hardware implementation. There are also researches which aimed to decrease the required memory [31]. The main idea to speed up maze running algorithm is to limit the propagation area. As maze running algorithm guarantees to find the shortest length path while line search guarantees to find the path with minimum number of bends, some researchers [26] combined both these two algorithms together to compromise between wire length and number of bends.

Watanabe et al. [26] have combined maze running algorithm (minimum length) and

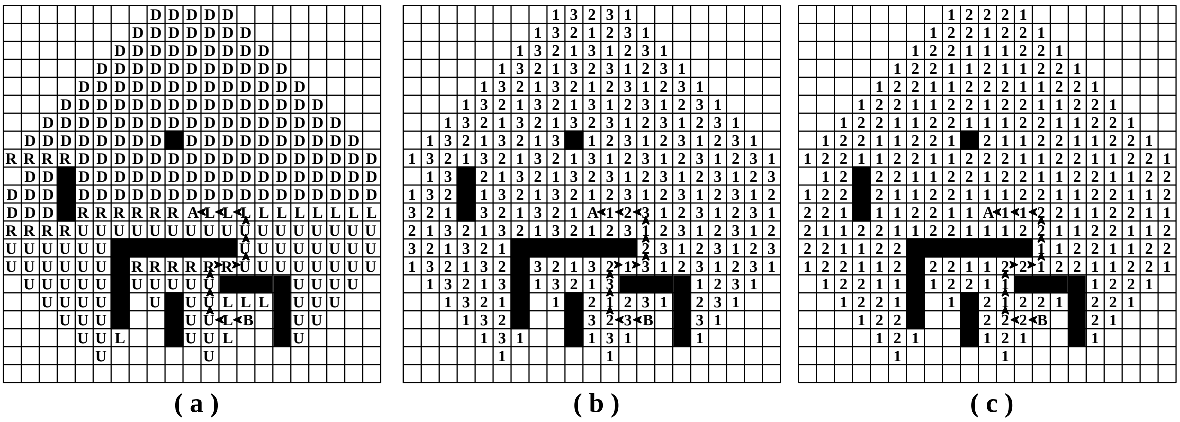


Figure 3.8: Decreasing the required memory.

line search algorithm (minimum bends) by introducing the concept of the expansion distance D_{ex}. This algorithm differs from the original Lee algorithm in that the expansion distance is not limited to unity but can take any value D_{ex} for each step. When D_{ex} is set to unity, it is equivalent to original Lee algorithm (Figure 3.7a). When D_{ex} is set to infinity, it functions as the line search algorithm (Figure 3.7b). If D_{ex} is greater than one and less than infinity, it gives a sub-optimal path in terms of both the wire length and the number of bends (Figure 3.7c).

3.3.1 Decreasing the required memory

The amount of memory space needed by maze running algorithm is a real problem. A unit of memory space is needed for each cell (grid point) to store the wave expansion label. If the grid dimensions is 4000×4000 and the expected wire length is up to 8000, then the required memory space is 16 million unit per layer and each unit must be 13 bits. Soukup [31] uses two heaps and one stack instead of the grid map. Some other labeling schemes rather than the ascending one shown in Figure 3.1 have been introduced to decrease the number of bits needed for each cell. Some of these schemes are:

1. The adjacent free cells are labeled by Left, Right, Up, or Down, which indicate the direction of wave propagation as shown in Figure 3.8a.

2. The sequence: 1,2,3,1,2,3,... is exploited for labeling as shown in Figure 3.8b.

3. Akers [32] introduces the sequence: 1,1,2,2,1,1,2,2,... for labeling as shown in Figure 3.8c.

Among these coding schemes the last one proposed by Akers is most economic. Since each cell will be in one of just four states; "empty", "occupied", or labeled by "1" or "2", two bits will suffice to proceed the single layer Lee algorithm. The first two coding schemes need three bits for the same information, but are almost equally efficient for some variation of Lee algorithm [22].

3.3.2 Decreasing the computation time

Maze running algorithm is time consuming. Many methods have been proposed to speed up maze running algorithms by decreasing the propagation area as follows:

1. Two terminals propagations: Starting the propagation from the two terminals together will decrease the propagation area as shown in Figure 3.9a.

2. Framing: Making an artificial rectangular frame containing the net terminals, and do not allow any propagation outside this frame as shown in Figure 3.9b.

3. Different weight: make the weight of the cells in the direction toward the target less than the weight of the cells in the direction away from the target. This will accelerate the propagation toward the target. (Figure 3.9c).

4. Make the weight of the cells in the direction of the target equal zero as shown in Figure 3.9d.

5. Combine maze with line search [30]. A line segment starting from the source is initially extended toward the target, and the cells on the line segment are

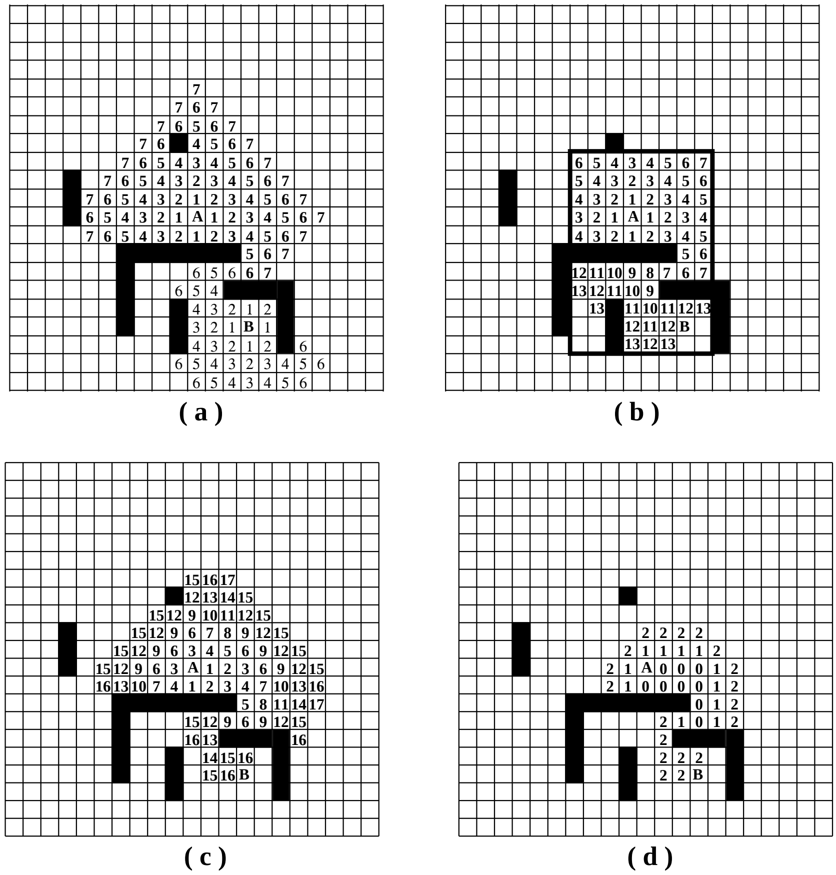


Figure 3.9: Fast maze.

searched first. When the line hits an obstacle, the Lee-type wave propagation is used to search around the obstacle. Once a cell approaching the target is found, another line segment starting from there is extended again toward the target. This algorithm guarantee finding a path whenever it exists, but the generated path is not always shortest.

3.4 Global Routing

With the increasing complexity of the routing problems dealing with thousands of nets, the large problems should be decomposed into two processes: global and detailed. In

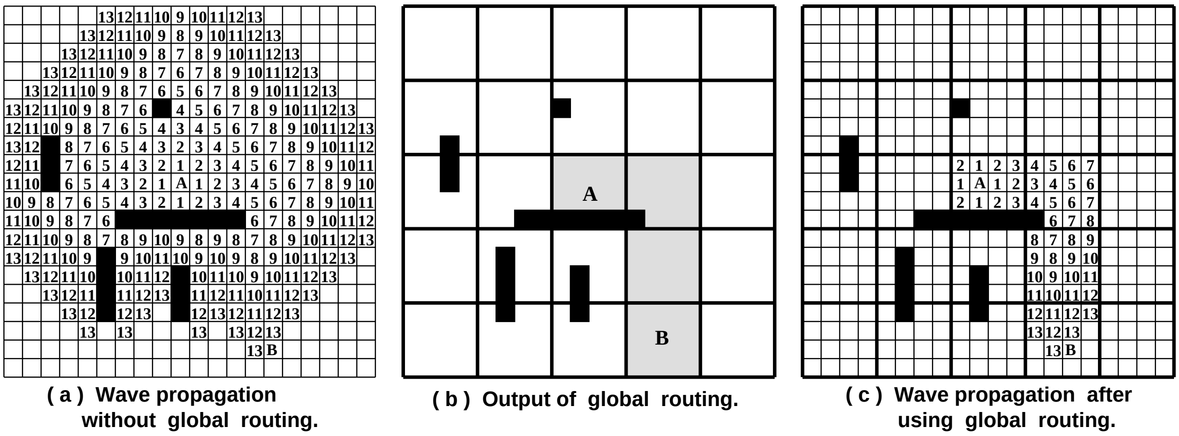
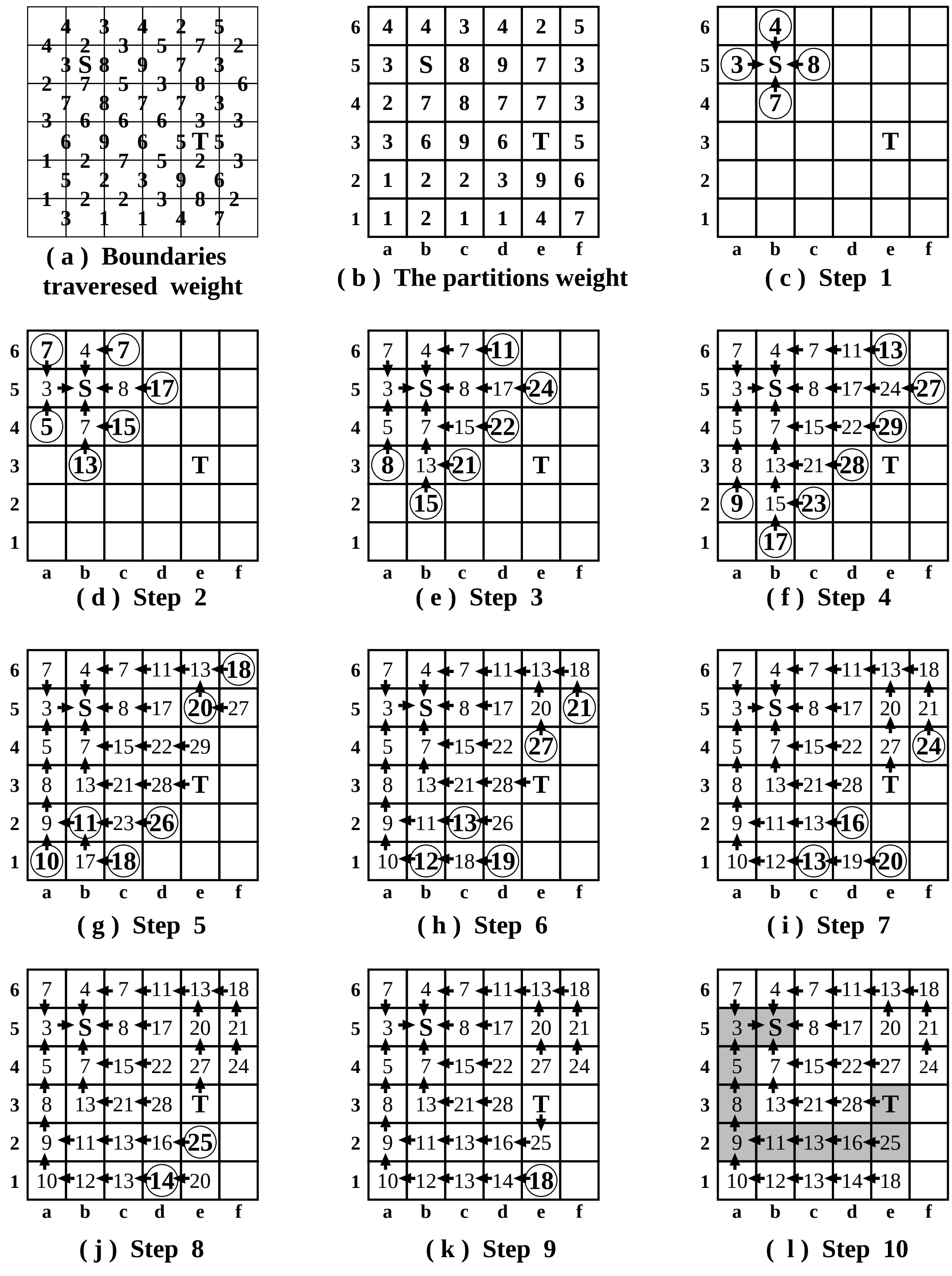


Figure 3.10: Using global routing decreases the propagation area.

the global routing (it is also called loose routing), the whole routing area is divided into partitions, and it is required to determine in which partitions each net will be routed. The exact path for each net within the partitions is found in the detailed routing[1]. Using global routing as a preliminary step decreases also the propagation area, and this will speed up the computation time. Figure 3.10 shows the effect of using global routing as a preliminary step on the propagation area. It could be seen from comparing Figures 3.10a and 3.10c that using global routing significantly decreased the propagation area.

It should be mentioned that 100% assignments in global routing do not always guarantee 100% interconnection of all specified nets. The final routing completion needs to be addressed again at the detailed routing step [33].

Since global routing is also an NP complete problem, a variety of heuristics have been proposed to solve the problem. Specifically, we have global routers based on graph search methods [4] [30], hierarchical decomposition techniques [34] [35] [36], and iterative improvement (for standard cells) [37] [38]. Here, only global routing based on the graph search will be explained in detailed as we use this approach in our router.



(a) Boundaries traveresed weight

(b) The partitions weight

(c) Step 1

(d) Step 2

(e) Step 3

(f) Step 4

(g) Step 5

(h) Step 6

(i) Step 7

(j) Step 8

(k) Step 9

(l) Step 10

Figure 3.11: Wave expansion on a weighted grid.

3.4.1 Graph search

The most common approach based on graph search for global routing is to route nets sequentially (one net at a time) using the Lee-Moore grid expansion algorithm and its modifications. It is not possible to route all the nets with the shortest connection paths because they compete with each other for paths to be used. Different costs (weights) are assigned to the cells or to the boundaries of the cells [25] as shown in Figure 3.11. The weight depends on congestion due to nets previously routed. Hence the algorithm for routing should find the cheapest path and not the shortest one.

The cheapest path running algorithm starts by assigning each empty cell adjacent to the source (S) a value equal to its weight (Step 1 in Figure 3.11). Each such cell is marked as a *latest* cell. At each step from here on, each latest cell C (with value V) is examined, and every empty cell (with weight W) adjacent to C receives a value V+W and itself serves as a latest cell for the next step. If a cell adjacent to C already has a value greater than V+W, then the old value is replaced by V+W and serves as a new latest cell. Otherwise the value of the adjacent cell is left as it stands. Figure 3.11 shows this procedure where latest cells are indicated by bold numbers inside circles. It should be noted here that some old latest cells may become new latest cells again if their values (total cost) are decreased as which happen to cell 5e in Step 5 in Figure 3.11.

Even if the target (T) is reached (step 5 in Figure 3.11), this does not always mean that the desired path has been found because it is possible that the old latest cell will receive a smaller value later. This implies that an already generated path may be overridden by a new one with less total weight. This happens in both Steps 7 and 9 in Figure 3.11 as a smaller cost is assigned to the target. For that reason, the wave propagation must be continue until there is no latest cell with value less than the value of the target (Figure 3.11). Another way is to use a priority queue for the set of the latest cells. That is meant that to sort the latest cells according to their weight, and expand the latest cell with the minimum weight.

Nair et al. have proposed a global routing algorithm based on the preceding algorithm that tries to alleviate the problems associated with the ordering of nets in the graph search approach. The effectiveness of the algorithm depends on the accuracy of the cost estimates at the various boundaries. Some sophistacted cost computation methods are used in this algorithm, which consist of four components:(1) the demand measure which characterizes the requirement of all nets that are yet to be wired, (2) the supply measure which estimates the remaining number of tracks left after a certain number of nets have been routed, (3) the cost of traversal which takes into account the physical complexity of the cell, and (4) the cost of placing vias on adjacent intersections [1].

For multi terminal nets, the source cell, the destination cell with the least cost, and all the cells that are on the backtrace path become additional source cells for the next forward propagation iteration.

3.4.2 Hierarchical decomposition

Two schemes have been proposed for hierarchical decomposition technique: the top-down [34] and the bottom-up [35]. In the top-down scheme, the global routing problem is divided into 2×2 subregions and the routing tasks are solved recursively. As one proceeds downwards, one obtains a better focus of all connections on each level of hierarchy. As one reaches the last step, one gets a clear view of the complete problem. In the bottom-top approach, parts of connections completed at the previous level are merged into larger chunks, and this process continues until all possible connections are made. In both cases, decisions on each level of hierarchy are made independently, i.e., without the knowledge of how they may affect the succeeding stages. The local decisions made at each step may result in inability to find final solution, even if one exists.

3.4.3 Iterative improvement

The algorithm for global routing using iterative improvement [37] [38] considers a subset of possibilities for routing nets (L-shaped and a few Z-shaped layouts) and routes multiples wires simultaneously. The run time of this algorithm is quite low, but the routing quality is not as good [1].

Chapter 4

Parallel algorithms for short nets

In this chapter, we introduce two new parallel algorithms based on maze running algorithm and suitable for routing small size layout (routing area) problems. Each of them divides the routing area by a new technique which is suitable for mesh network computers. These algorithms have high degree of parallelism inside and obtain high speedup specially for short nets. Short net is a net which has short Manhattan distance between its terminals as compared with the dimensions of the whole routing area. In the first algorithm, moving boundaries are used in dividing the routing area. The second algorithm is divided into two phases. In the first phase, we rotate the areas assigned to the processors to route all short nets. In the second phase we use the competing processors algorithm [10] to route the remaining nets. We implemented our two algorithms and two other algorithms which have been introduced by Takahashi et al. [11] [10] on the AP1000 computer. The results and comparisons between the four algorithms are also included.

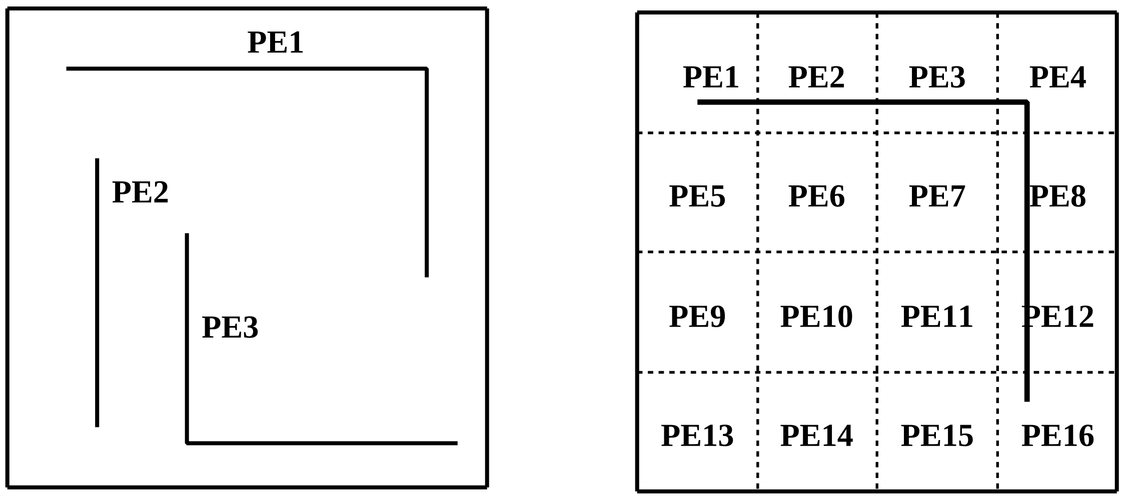
4.1 Introduction

The basic problems of automated wire routing are the long computation time and large memory size required. Although Lee's algorithm [4] which is known as Maze running was the first algorithm to solve the automated wire routing problem (1961), it is still used until now because it has a high degree of flexibility. Many researches [29] [30] modified the original maze running to make it faster, but the improvement in the speed was not satisfactory. There are also researches which aimed to decrease the required memory [31]. Some other researches [6] tried to improve the speed by using hardware implementation of maze running algorithm. However, hardware accelerators are not always cost effective because these machines will soon become obsolete according to the fast changing in technologies. Recently, several researches have tried to speed up the routing problem by using general purpose parallel computers.

Parallel computers are used to overcome long computation time and large memory size required for automated wire routing. Parallelization could be obtained in both nets and space (routing area). In the former, nets are divided among the PEs (Figure 4.1a) so that every PE will be responsible for routing some nets [10]. In the later, the routing area is divided into partitions and one or more partitions are assigned to each PE. Several PEs are sharing in finding a path for a net so that each PE from these PEs finds the part of this path within its assigned area(Figure 4.1b). To increase the parallelization, some researches merge both of these two ways together [7] [14].

This chapter introduces two parallel algorithms based on maze running algorithm and suitable for routing small size routing area problems. Each of them divides the routing area by a new technique which is suitable for mesh network computers. These algorithms have high degree of parallelism inside and obtain high speedup specially if the Manhattan distances between the terminals of the nets are short in comparing with the whole routing area. We will call these nets as short nets. In the first algorithm, moving boundaries are used in dividing the routing space. The second algorithm is divided into two phases. In the first phase, we rotate the areas assigned to the processors to route all short nets. In the second phase we use the competing processors algorithm [10] to route the remaining nets.

We implemented our two algorithms and two other algorithms which have been introduced by Takahashi et al [10] [11] on the AP1000 computer(message passing,



(a) Parallelization between nets (b) Parallelization in the routing area.

Figure 4.1: Nets parallelization and space parallelization.

torus network, MIMD parallel computer) [21]. The results and comparisons between the four algorithms are also included in this chapter.

4.2 Moving Boundaries Algorithm (MBA)

This algorithm could be considered as a modification of the hierarchical domain partitioning algorithm **HDPA** introduced by Takahashi et al [11]. In MBA algorithm, the host computer broadcasts a list of all net terminal locations and the dimensions of the routing area to all the PEs. The routing area is divided into equal partitions, and one partition is assigned to each PE. Number of partitions should equal to the number of PEs. As an example, if the grid dimensions is 256×256 and total number of PEs is 64 (consider it as 8×8 PEs), then the partition area assigned to each PE is 32×32 (Figure 4.2 step 1). Every PE selects the nets which lie inside its assigned partition from the list of all nets, sorts them in an ascending order according to their Manhattan distances, and routes them one after the other within its assigned partition. There is no possibility of any confusion between different PEs.

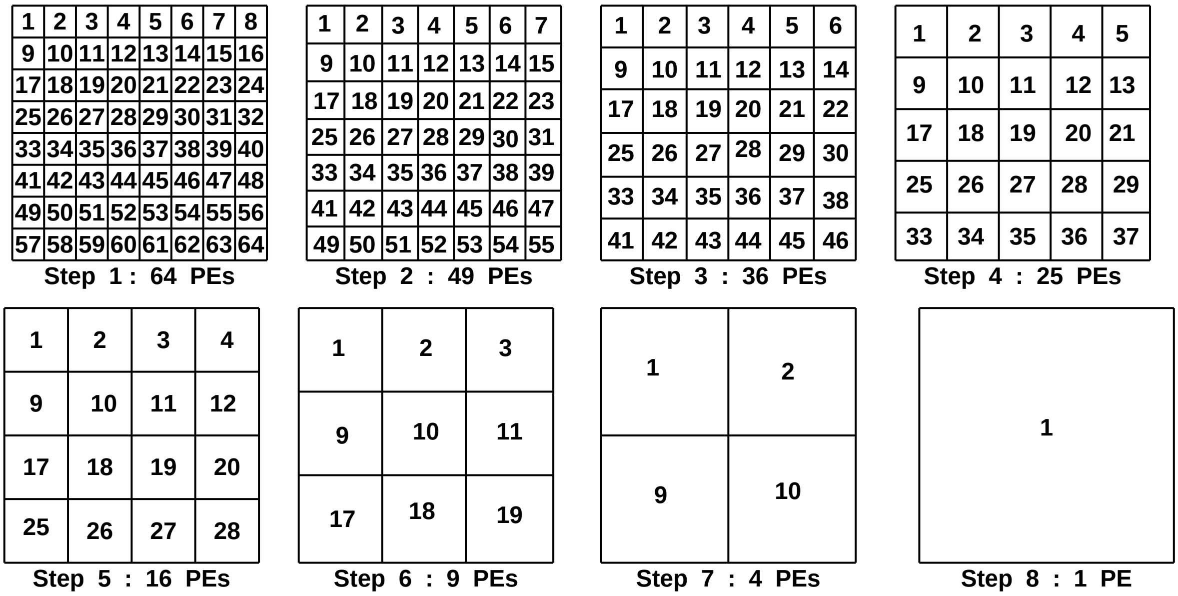


Figure 4.2: Partitions in moving boundaries algorithm (MBA).

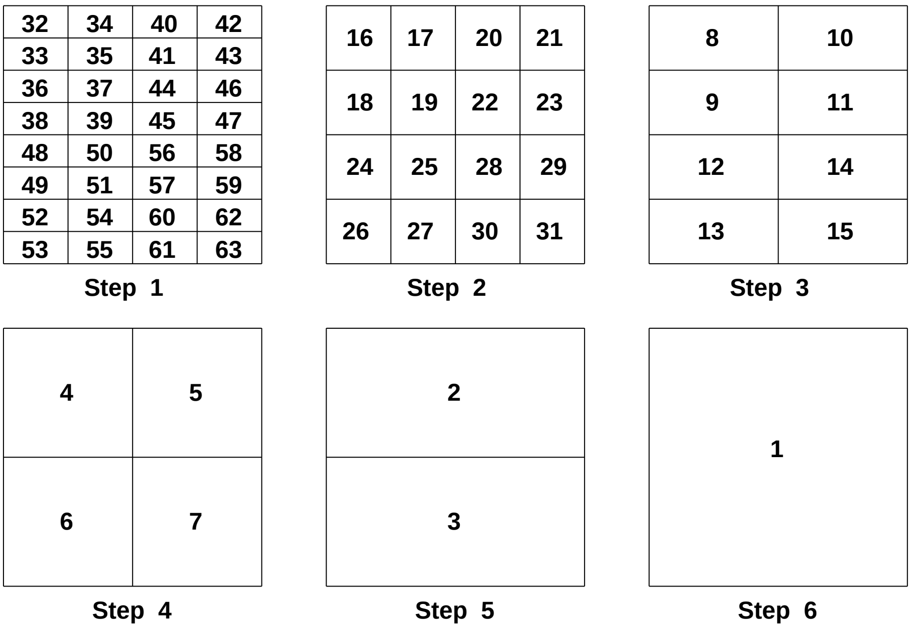


Figure 4.3: Partitions in hierarchically domain partitioning algorithm (HDPA).

After finishing this step, the number of active PEs is decreased to 49 PEs (7×7) as shown in Figure 4.2 step 2. All the boundaries between the partitions will be changed from the previous (first) step. Every PE calculates its new boundaries, sends the leaving part of its bit map, which contains the nets already connected, to its left and up neighbor PEs, receives the taking part of the bit map from its right and down neighbor PEs, and adds the taking part of the bit map to the remainder part of its old bit map to obtain the new bitmap which is 36×36 or 37×37 in this example. Then every active PE selects again the unrouted nets which lie in its new area, sorts and routes them as in the first step.

In the third step, the number of active PEs will be decreased again to 36 PEs(6×6) as in Figure 4.2 step 3. The PEs will again calculate the new boundaries, send and receive data, select, sort,and route the unrouted nets. And so on until the last step where one PE will be responsible for all the routing space. Of course we may delete some steps, so we can make the steps, for example, as $8 \times 8 \rightarrow 7 \times 7 \rightarrow 5 \times 5 \rightarrow 3 \times 3 \rightarrow 2 \times 2 \rightarrow 1 \times 1$. We can also jump some other steps.

The major difference between the MBA and the HDPA is that the boundaries are moved in MBA though they are fixed in HDPA (Figure 4.3). For example, if we have 256×256 grid and 64 PEs, then the boundaries for different steps are as shown in Tables 4.1 and 4.2. We can see that in the HDPA the boundary 128 exists from the beginning until the last step. Then at the last step one PE will be responsible for routing the long nets and also short nets which lie around this boundary. In the MBA, short nets are routed by more than one PE in some previous steps.

In other words, MBA has an advantage that no short nets will remain without routing until the end. So, the last steps, where only few PEs are active, will be aimed to route long nets only. But in HDPA, short nets around the boundaries will remain without routing until the end. An example to explain this feature is shown in Figures 4.4 and 4.5. In case of of HDPA (Figure 4.4), both short nets (A, B) and long nets (C) are connected in the last step (step4) using only one PE. In case of MBA (Figure 4.5), short nets (A,B) are connected in parallel in step2 using 2 PEs, and only one long net

Table 4.1: Grid partitions in hierarchically domain partitioning algorithm (HDPA)

	PEs	area	x boundaries	y boundaries
step 1	32	64×32	64,128,192	32,64,96,128 160,192,224
step 2	16	64×64	64,128,192	64,128,192
step 3	8	128×64	128	64,128,192
step 4	4	128×128	128	128
step 5	2	256×128		128
step 6	1	256×256		

Table 4.2: Grid partitions by moving boundaries algorithm (MBA)

	PEs	area	x boundaries	y boundaries
step 1	64	32×32	32,64,96,128, 160,192,224	32,64,96,128 160,192,224
step 2	49	36×36	36,73,109 146,182,219	36,192,224 146,182,219
step 3	36	42×42	42,85,128 170,213	42,85,128 170,213
step 4	25	51×51	51,102,153, 204	51,102,153, 204
step 5	16	64×64	64,128,192	64,128,192
step 6	9	85×85	85,170	85,170
step 7	4	128×128	128	128
step 8	1	256×256		

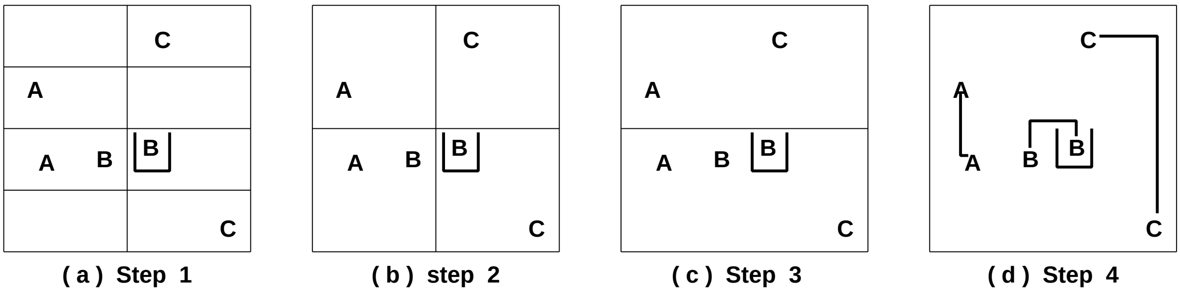


Figure 4.4: Routing nets in hierarchically domain partitioning algorithm (HDPA).

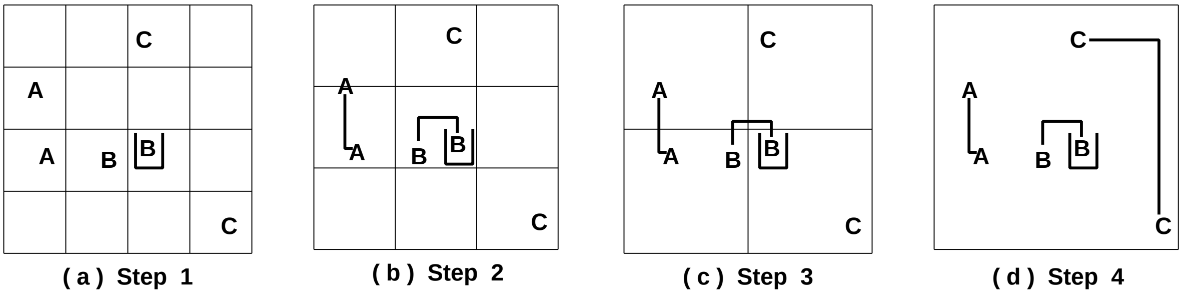


Figure 4.5: Routing nets in moving boundaries algorithm (MBA).

(C) will be routed at the last step using one PE.

MBA get good results if the average Manhattan distance of the nets is short compared with the whole routing space because most of the nets will be connected in the first steps where most of PEs are active, while small number of long nets will stay without routing until the end. On the other hand, the parallelization will be decreased if the average Manhattan distance is long compared with the routing space. This is because only few PEs will be used to route a large number of long nets which require longer time for routing.

4.3 Rotating Areas Algorithm (RAA)

This algorithm consists of two major phases. The first phase is aimed to route short nets using only few messages by rotating the area assigned to each PE. The remaining nets will be routed in the second phase using the competing processing algorithm

(**CPA**) which have been introduced by Takahashi et al [10].

4.3.1 First phase

We define X_{diff} for a net as the maximum distance between its terminals in the X direction (Y_{diff} in Y direction). If the number of PEs is N^2, and the routing area is a single layer grid with dimensions are $L \times L$, then this phase guarantees to try to find a path for all nets which have both X_{diff} and Y_{diff} less than S, where $S = L/(2N+1)$. This phase may also deal with nets which have X_{diff} and/or Y_{diff} greater than S but less than $2S$ according to the locations of the net terminals within the partitions.

In this phase, the host computer broadcasts the locations of all net terminals and the grid dimensions to the PEs as in the moving boundaries algorithm. The grid is divided also by the total number of PEs, but here, we do not divide all the grid. There is a part of the grid which is not assigned to any PE as shown by the shaded area in Figure 4.6a. So, every PE is responsible for an area of $d^2 = ((L-S)/N)^2$ only (instead of $(L/N)^2$ in MBA).

In the beginning, all the nets are labeled by 0 which means that these nets have not been routed yet. Every PE selects the nets which lie in its assigned area, sorts them in an ascending order according to their Manhattan distances, and routes them one after the other within its assigned area. If all the terminals of any net are located in a PE area, then this PE tries to find a path for this net. The net will be labeled by 1 if the PE succeeds to find a path, and it will be labeled by 2 or 3 if the path could not be found at all (like net B in Figure 4.7) or not found but may be found later because this net is near any boundary (like net A in Figure 4.7).

After finishing this step, the shaded area is changed, and half of the area assigned to every PE is changed (Figure 4.6b). Every PE sends the data of half of its previously assigned area and the net list to the left neighbor and receives another data from the right neighbor. Every PE selects again nets (labeled by 0 or 3) which lie in its area, sorts and routes them.

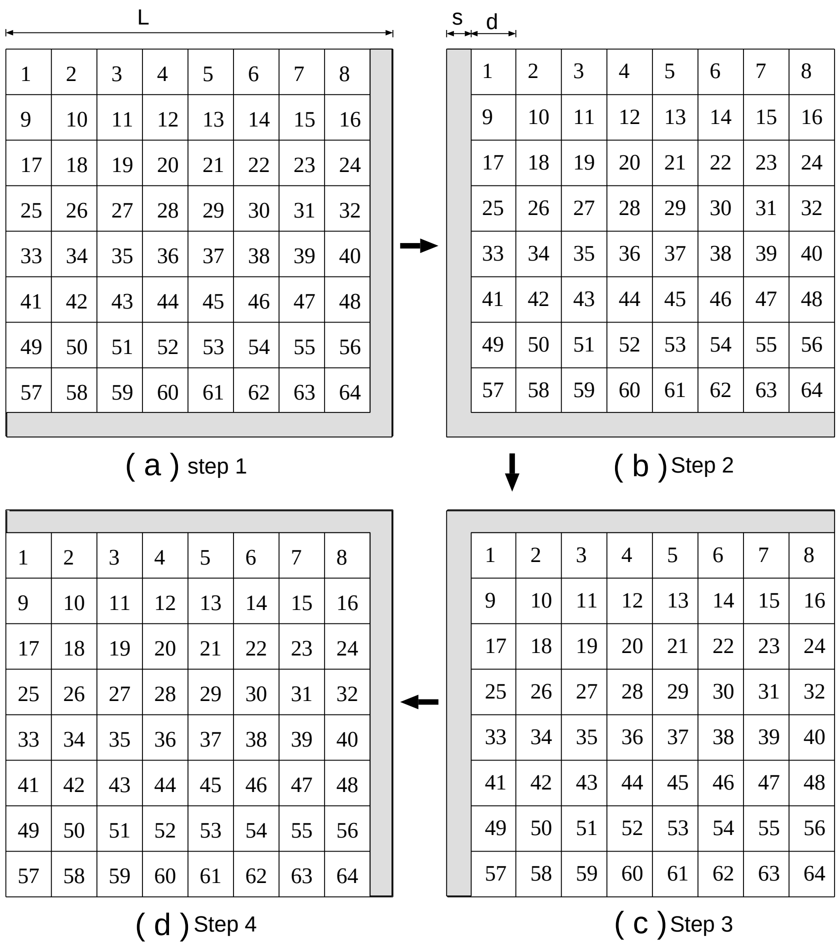


Figure 4.6: The first phase of rotating areas algorithm (RAA).

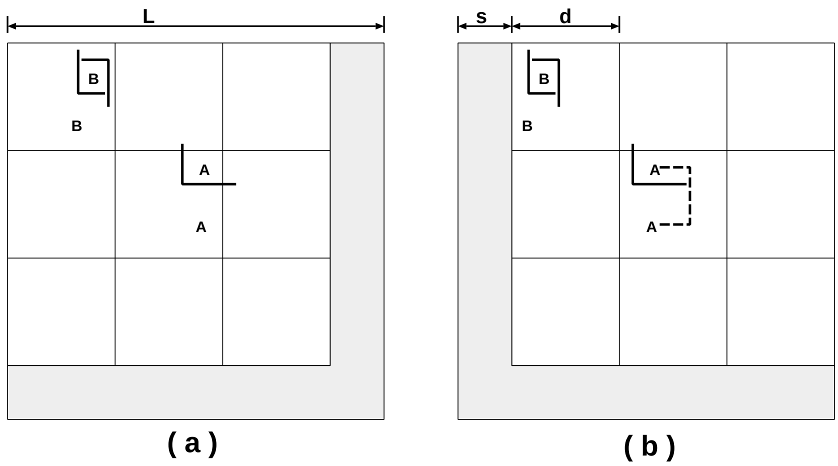


Figure 4.7: Some unrouted nets may be routed in next steps.

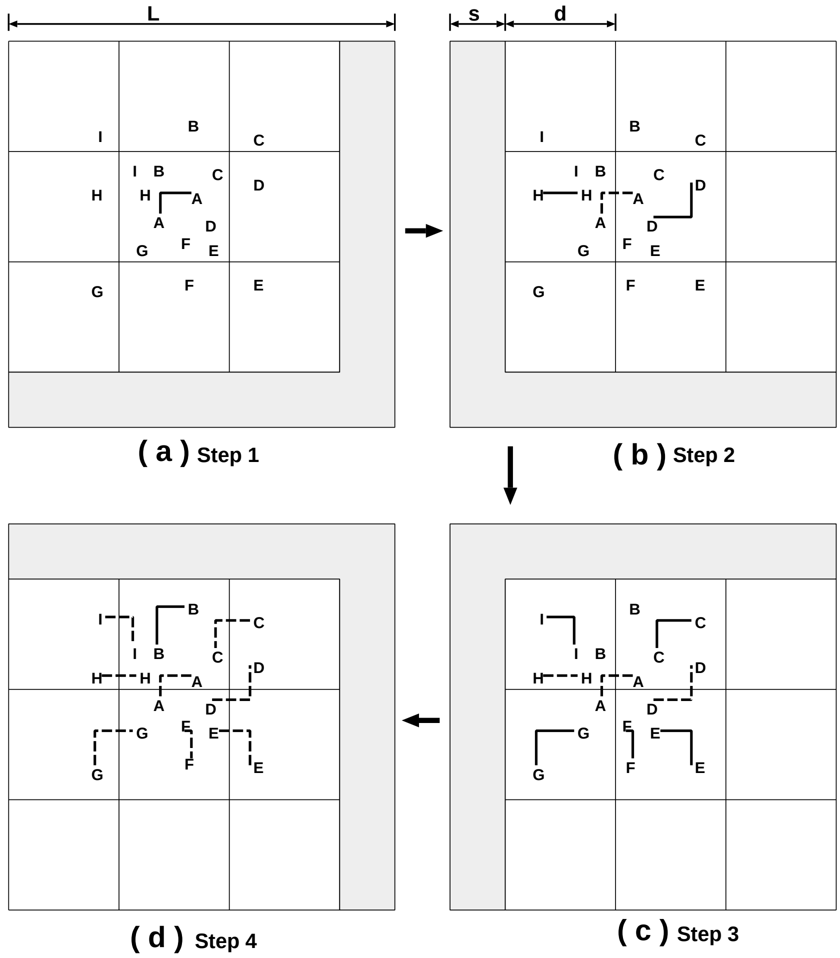


Figure 4.8: Example of routing short nets at the first phase of rotating area algorithm.

The shaded area will be changed again (Figures 4.6c and 4.6d). At the end of this phase, all the PEs send their data (the paths of nets) and the net list to the host computer.

By this complete rotation, we ensure that all short nets can be connected regardless of their locations. Figure 4.8 shows an example to explain this property. In step 1, only net A is routed and all the other nets have not been routed yet. In step 2, both nets D and H are routed while nets C,E,G and I are routed in step 3. Net B will be routed in step 4.

The communication cost of MBA and the first phase of RAA are very low as only

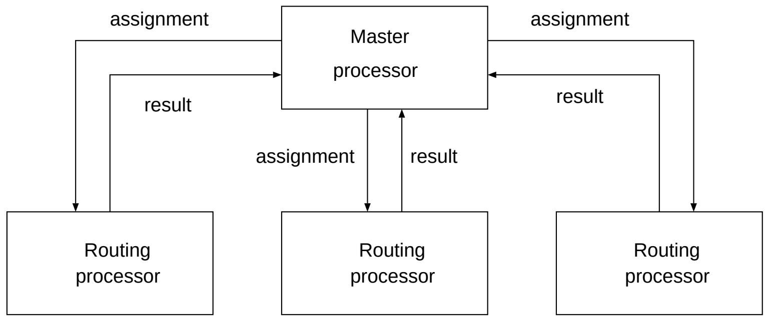


Figure 4.9: The second phase of rotating area algorithm.

few messages have been used to route all nets. Moreover, all these messages are sent to the neighbor PEs in case of mesh or torus network, parallel computer.

4.3.2 Second phase

In the second phase of this algorithm, we use the competing processors algorithm **CPA** [10]. In CPA, one processor or the host computer is used as a master and all the other PEs are used as slaves for routing nets in a net parallelization manner (Figure 4.9). The master PE broadcasts the bit map of the whole grid to all the slaves and assigns a net (which has a label 0 or 3) from the net list to every slave. Every slave PE tries to find a path for its assigned net, and sends this path to the master PE. The master PE verifies if each of these received paths intersects with any other previously accepted paths. If it intersects then the master PE will send it again to the sender slave PE with the latest accepted paths. But if it does not intersect then the master PE labels this net by 1, adds its path to the list of accepted paths, sends the latest accepted paths and assigns another net to the sender slave PE. These steps will be repeated for all nets labeled by 0 or 3.

By using two phases in RAA, all short nets are routed in the first phase using few messages, while only long nets are routed in the second phase. This has two main advantages over CPA which uses only the second phase to route both short and long nets. These advantages are:

1. If we use the second phase to route short nets which require short routing time, then many messages will arrive to the master PE at the same time. The master PE will not be able to serve all the routing PEs on time especially if the number of PEs is large. The routing PEs have to wait for a long time for the master PE. But this hardly happen for long nets which required longer time to be routed.

2. It decreases total number of messages from

   $2\times$*(number of nets + number of reroute nets)*

   to be

   $2\ \times$*(number of long nets + number of reroute nets)*

   and of course the number of reroute nets will be decreased also.

The first phase can be used alone under the restriction that the maximum Manhattan distance is short as compared with the grid dimensions. For example, if the maximum of X_{diff} and Y_{diff} is shorter than 500 in a grid dimension 10000×10000 and the number of PEs is 64, then every PE will be responsible for area of 1176×1176 only. By this, we can decrease the required memory for each PE from 10000×10000 to 1176×1176. On the other hand, ignoring the second phase may decrease the successful routing ratio as some nets (which are labeled by 3) may be routed during the second phase.

All the algorithms introduced in this chapter (HDPA, MBA, RAA, and CPA) could be used only for small size routing space problems. This is because these algorithms do not divide the required memory among the PEs. In certain step in each of all these algorithms, one PE at least must have a copy of the whole routing grid in its local memory.

4.4 Results

Several experiments have been done and many results have been taken to see the performance of the algorithms introduced in this chapter. Now, we will discus some

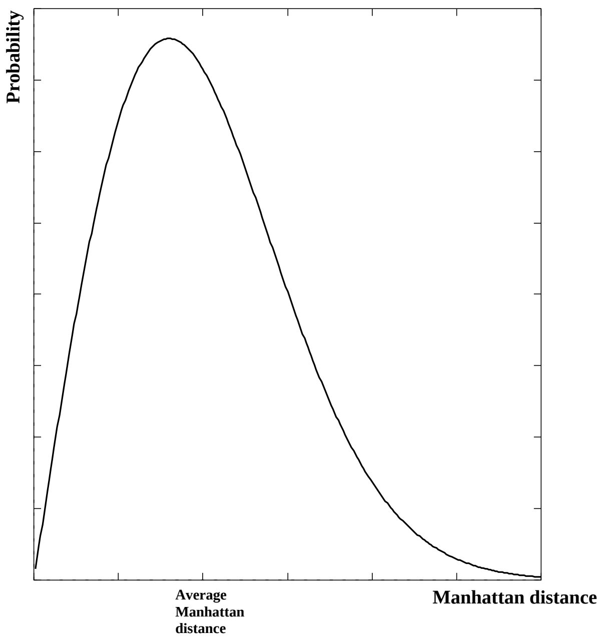


Figure 4.10: Rayleigh probability function.

of these results. These results were taken on a single layer grid using test data (nets) as input to our program. The test data generator program output random x and y coordinates for network terminals. The Manhattan distances of the nets obey the Rayleigh probability function shown in Figure 4.10. If there is a two terminals net with coordinates (X_1, Y_1) and (X_2, Y_2), then

$$Manhattan\ distance = abs(X_1 - X_2) + abs(Y_1 - Y_2)$$

We implement our new two algorithms (MBA and RAA) with that of two other algorithms (HDPA and CPA) introduced by Takahashi [10] [11] to compare their performances. Four programs have been implemented on the AP1000 parallel computer using C language. Although both HDPA and CPA are originally used line search to route the nets, we use maze running algorithm in all the four programs to be able to get an accurate comparison between the parallelism inside the four programs. We obtained better successful routing ratio by using maze running algorithm than the routing ratio reported in [10] using line search as shown in Table 4.3.

Table 4.3: Routing ratio for CPA on 256×256 single layer grid with average Manhattan distance 20 using 64 PEs.

	Number of nets					
	100	200	400	1000	2000	3000
Using maze	100%	100%	98%	83%	64%	53%
Using line search [10]	100%	100%	97%	81%	61%	51%

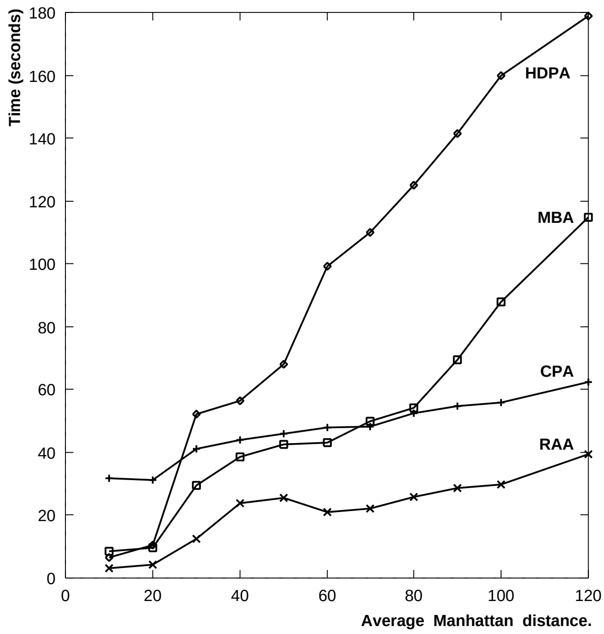
The following results for all the four programs are taken on a single layer grid with dimensions 1000×1000 using maze running algorithm.

4.4.1 Effect of changing average Manhattan distance

The average Manhattan distance of the nets has a big effect on both HDPA and MBA. For short average Manhattan distance, most of the nets will be routed in the first steps using most of the PEs. For long average Manhattan distance, most of the nets will be routed in the last steps using less number of PEs. This average Manhattan distance has less effect on both CPA and RAA as shown in Figure 4.11. In this Figure, we measure the routing time for different average Manhattan distance for 4000 nets. As we can see MBA is faster than HDPA for almost all different Manhattan distances and also MBA is faster than CPA for short average Manhattan distance less than 70.

4.4.2 Effect of changing nets number

In order to know the effect of changing the number of nets on the different algorithms, we measure the time for different nets number with a fixed average Manhattan distance (for example = 20). Nets number has a big effect on CPA because it increases the number of messages. If the average Manhattan distance is small as in this case, then the routing PEs will route nets in a short time which will make overhead at the master PE because many messages will arrive at the same time. Increasing nets number does



HDPA : Hierarchical domain partitioning algorithm
CPA : Competing processors algorithm
MBA : Moving bouundary algorithm
RAA : Rotating area algorithm

Figure 4.11: Time against average Manhattan distance for 4000 nets.

not increase the time so much in HDPA, MBA and RAA because most of these nets will be routed at first step by all the PEs and the communication cost is independent of nets number for these three algorithms. Figure 4.12 shows time against nets number for average Manhattan distance equal 20.

4.4.3 Effect of changing both the nets number and average Manhattan distance together

In Figure 4.13 we change both the average Manhattan distance and number of nets so that we keep the total nets lengths (average Manhattan distance $\times$ nets number) constant equal to 200,000. For example, if the average Manhattan distance is 10 then the nets number will be 200,000/10 =20,000. As could be seen CPA takes a very long

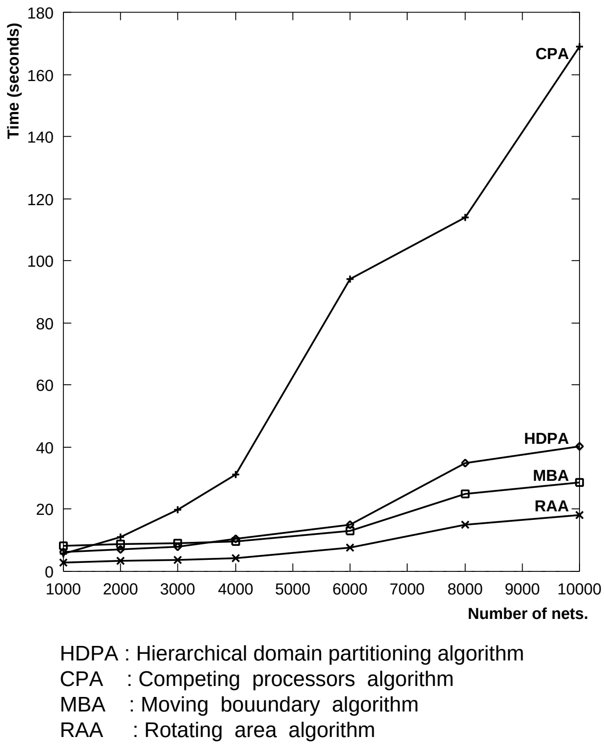


Figure 4.12: Time against nets number for average Manhattan distance equal 20.

time for routing large number of short nets and it takes shorter time for less number of nets with longer Manhattan distances. This changing of both Manhattan distance and number of nets has a less effect on RAA which gives the best result among all the four algorithms.

4.4.4 Speedup

Figure 4.14 shows the speedup of the different algorithms when routing 8000 nets with average distance 20. The speedup is calculated by dividing the routing time using only one PE by the the routing time using different number of PEs. RAA has the best speedup among the four algorithms. MBA gives better speedup than both HDPA and CPA. Increasing the average Manhattan distance of the nets improves the speedup for CPA.

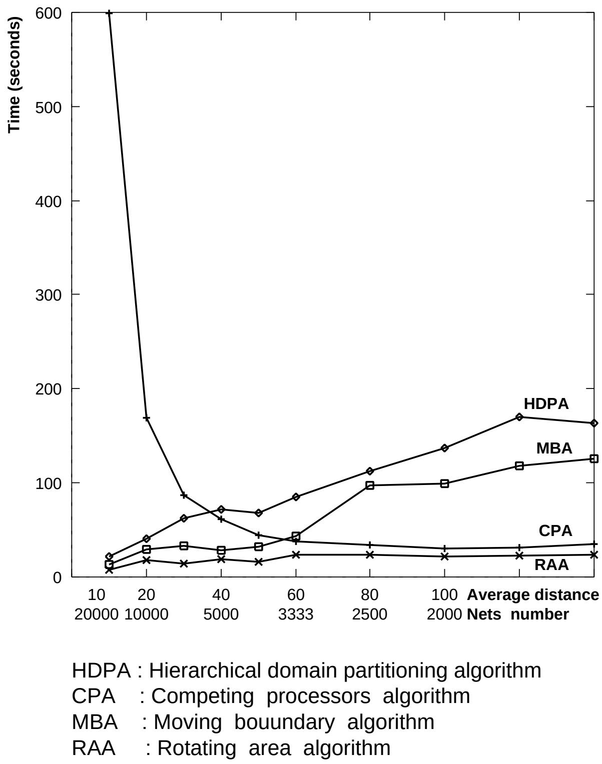


Figure 4.13: Time against nets number and average Manhattan distance with fixed total length.

4.5 Summary

In this chapter, two parallel routing algorithms for dividing a grid has been introduced. Both of these algorithms are focused to decrease the routing time especially for short nets. In the first algorithm (MBA), the whole grid area is firstly divided to N partitions and each of these partitions is assigned to a processor to route the nets in parallel. After that the whole grid is re-divided to M partitions such that each new boundary is different from the previous boundaries of the N partitions. The second algorithm (RAA) consists of 2 phases. In the first phase, we exploit the parallelism in the space by assigning a partition to each processor then we move every partition to route all the nets shorter than a certain Manhattan distance. In the second phase, we exploit the

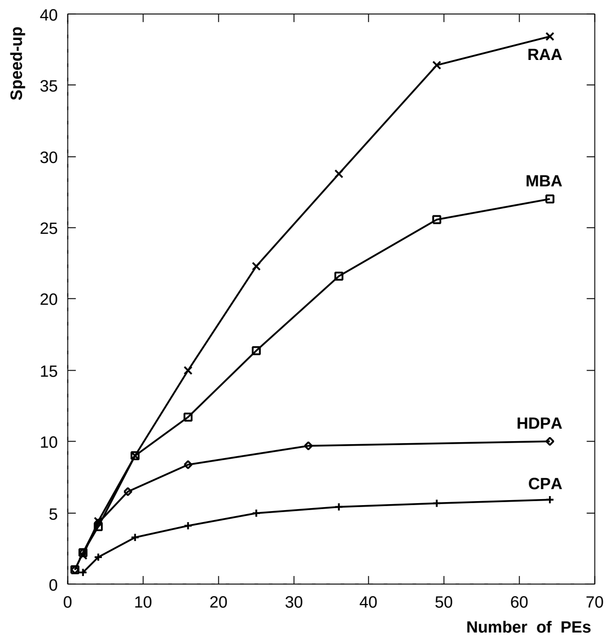


Figure 4.14: Speed up against number of PEs for 8000 nets with average distance 20.

parallelism in the number of nets using the competing processors algorithm (CPA).

Our new two algorithms (MBA,RAA) together with two other algorithms (HDPA,CPA) [10] [11] are implemented on the AP1000 computer (MIMD message passing parallel computer). The results proved that our new algorithms are faster than the other algorithms. The moving boundaries algorithm (MBA), which is very easy to be implemented, gives a good result specially when routing large number of nets with short average distance. The rotating areas algorithm (RAA) gives the best results among the four algorithms for different average net distances and different nets numbers.

Although these algorithms are implemented for a single layer grid, they can be easily extended to be used in multi-layers grid. But all the four algorithms (HDPA, MBA, RAA, and CPA) could be used only for small size routing space problems. That is because these algorithms do not divide the required memory among the PEs. In some steps in these algorithms, one PE at least must have a copy of the whole routing grid in its local memory. Other algorithms which can be used for a large size multi-layer grid will be explained in the next two chapters.

Chapter 5

Amon: A parallel slice wire routing algorithm

This chapter introduces a new parallel wire routing algorithm called Amon which can be used for large scale multi-layer layout. Amon consists of two processes: global and detailed. Both of these processes are done simultaneously in two stages pipeline. The PEs are divided into global PEs and detailed PEs. We use maze running algorithms and its modifications for both global and detailed routing. In the global routing, the nets are divided among the PEs to route them in parallel. In the detailed routing, to overcome the problem of the large memory size required, the whole routing area is divided among the detailed PEs. We divide the multi-layer grid into layers, divide each layer into slices, and assign one or more slice to each detailed PE. The detailed PEs route nets, which has a non overlapping global paths, in parallel within their own slices. Dividing grid into layers increases the parallelization between nets and decreases the number of vias used (higher routing quality). Dividing each layer into slices decreases the communication cost between PEs by using fewer messages.

5.1 Introduction

In this chapter, a new parallel wire routing algorithm, which can be used for large scale multi-layer grid, is introduced. This router consists of two steps: Global and Detailed. In the global routing, we use maze running algorithm with different costs. The nets are divided among the PEs to route them in parallel (nets parallelization). Parallel maze global routing using nets parallelization has a high degree of parallelism. On the other hand using large number of PEs for global routing may decrease the routing quality.

One of two methods can be used to allocate jobs to the PEs. The first method is to use all the PEs at first in the global routing until they finish this step, and then use all of them again for detailed routing. The advantage of this method is that it allows the PEs to make an overall optimization for global paths after finishing the global process and before starting detailed process. The detailed process always needs longer computation time than global process.

In the second method, the PEs are divided into global PEs and detailed PEs. Both the global and detailed processes are done simultaneously in two stages pipeline. This method has three main advantages:

1. It limits the bad effect of using large number of PEs on the routing quality of the parallel maze global process (which uses nets parallelization) as only few of these PEs will be assigned to the global process and all the other PEs will be assigned to the detailed process.

2. It allows an interaction between global and detailed routing processes as will be shown in the next chapter.

3. It simplifies the program for the PEs as each PE is responsible for either global or detailed process.

For these advantages, we prefer to use the second method. Both global and detailed are done in some kinds of master-slaves manner. One PE is used as a master and the other PEs are used as slaves for routing the nets. Figure 5.1 shows an example

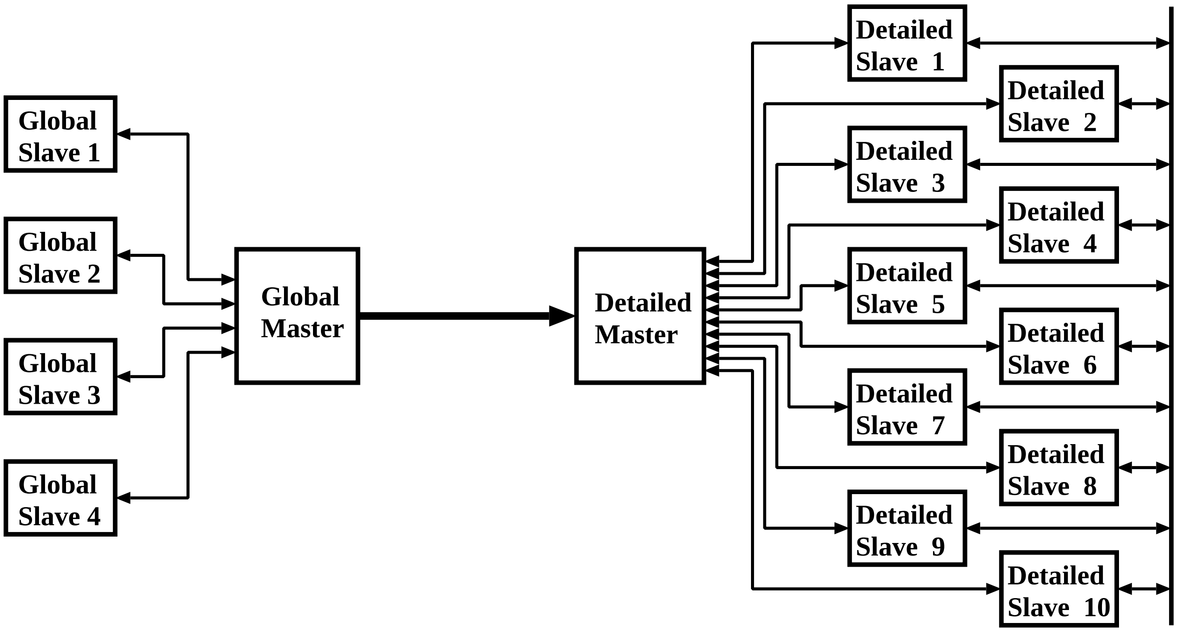


Figure 5.1: Allocate the PEs to the global process and detailed process.

of allocating 16 PEs to the global and detailed processes. In this example, 5 PEs are allocated to the global process (1 PE as global-master and 4 PEs as global-slaves), and 11 PEs are allocated to the detailed process (1 PE as detailed-master and 10 PEs as detailed-slaves).

In the detailed routing, the modified maze running algorithm with different costs is used to find the path for each net. To overcome the problem of the large memory size required for using maze running algorithm for detailed routing, the whole routing area is divided among the detailed PEs. The efficiency of parallelism depends on how to divide the whole routing space into partitions, and how to assign these partitions to the PEs. We divide the multi-layer grid into layers, divide each layer into slices, and assign one or more slice to each detailed PE. The detailed-master PE chooses the nets which have a non overlapping global paths, the detailed-slaves PEs route these nets in parallel within their own slices. Dividing grid into layers increases the parallelization between nets and decreases the number of vias used (higher routing quality). Dividing each layer into slices decreases the communication cost between PEs by using fewer messages [39].

Maze running algorithm has a serious disadvantage that previous routed net paths may prevent the other nets from being routed later. A new method for detailed routing to overcome this disadvantage will be introduced in the next chapter.

5.2 Global routing

In the global routing, the whole grid is divided into partitions, and it is required to determine in which partitions each net will be routed. This is done without knowing the exact path within each partition. We define *global path* for a net as the partitions in which this net will be routed inside. Global routing is used to decrease the required time for detailed routing, to improve the parallelization between nets, and to obtain better routing quality (shorter wire length and fewer vias).

We use maze running algorithm with different costs to find the global path for each net. The required memory to run maze algorithm for global routing is not so large because it is proportional with number of partitions instead of number of grid points as in the case of detailed routing. Nets parallelization could be used for global routing. In case of detailed routing, it is better to use space parallelization to reduce the required memory.

Every global PE has a copy of the cost of all the partitions in its own (local) memory. The cost mainly depends on the partition capacity (number of available tracks inside this partition). The global routing is done in a master-slaves manner using the competing processing algorithm (CPA)[10].

One global PE is used as a master (*global-master*) and the other global PEs are used as slaves (*global-slaves*). At the beginning all the nets are labeled by 0, which mean that no trails to route these nets is done yet. The global-master sends the capacities of all the partitions to the global-slaves, and assigns a net which has a label 0 to each global-slave.

Global-slaves use maze running algorithm (and its modifications) with different costs to find the least cost global path for the nets. The cost of each partition depends mainly on the number of available tracks (capacity) inside this partition. The global-master estimates these capacities at the beginning and update the capacities (by subtracting 1) of the partition located on global paths. More accurate way to calculate partition capacities will be introduced in the next chapter. Our global routing has a restricted X and Y direction (only X direction is allowed for odd layers and Y direction for even layers).

Each global-slave, after finding the least cost path of its net, sends this path to the global-master which verifies whether this path exceeds the capacity of any partition. If it exceeds then the master PE sends this net back to the sender global-slave with the latest accepted paths to reroute this net again after updating partition costs. Otherwise, the global-master labels this net by 1, sends its path to the detailed-master, adds this path to the latest accepted path list, updates the partition capacities, and sends another net with the latest accepted paths to the sender global-slave.

To decrease the communication cost between global-master and detailed-master, the global-master sends the paths of the completed nets in groups instead of one by one. The global-master stores the accepted paths in a list, and sends this list to the detailed-master after completing certain number of nets.

The algorithm for global-master is shown in Figure 5.2. The global-master uses an array of pointers which indicates the address of the last data (global paths) which was sent to every global-slave. The global-master uses this array to detect the latest accepted paths which must be sent to every PE. Figure 5.3 shows an example of this array in case that the number of global-slaves is equal to four.

Although competing processors algorithm (CPA) was originally introduced for detailed routing [10], CPA may suffer from large number of rerouted nets when it is used to perform detailed routing [14]. In the detailed routing, if more than one PE use the same grid point to route different nets at the same time, then all of these nets except the first received one have to be rerouted again. This is because each grid point

```
   Sort the nets in ascending order according to their Manhattan distance.
   Label all the nets by 0.
   For each global-slaves do
       Send the capacities of all the partitions.
       Choose a net which has a label 0
       Send the net identification to the global-slave.
   end for
loop:
   Wait until any message is received from any global-slave.
   If the global-slave find a path for its net.
     then
         If this path exceeds any of the partitions capacities.
           then
                Send the latest accepted paths to the sender global-slave.
                Send the same net identification to this sender global-slave.
                Update the pointer related to this sender global-slave.
                Goto loop.
           else
               label this net by 1.
               Add this net path to the list of latest accepted paths.
               Update the pointer of the last accepted path.
               Decrease the capacities of the partitions used by this path.
               Send this path to the detailed-master PE.
         end if
     else  label this net by 2.
   end if
   If there is any other nets label by 0
     then
         Send the latest accepted paths to this sender global-slave.
         Update the pointer related to this sender global-slave.
         Choose one net label by 0, label this net by 1.
         Send this new net identification to the sender global-slave.
     else  Send a message to this sender global-slave to end its job
   end if
   If all the global-slaves end their jobs
     then   end
     else   goto loop
```

Figure 5.2: Algorithm for global-master.

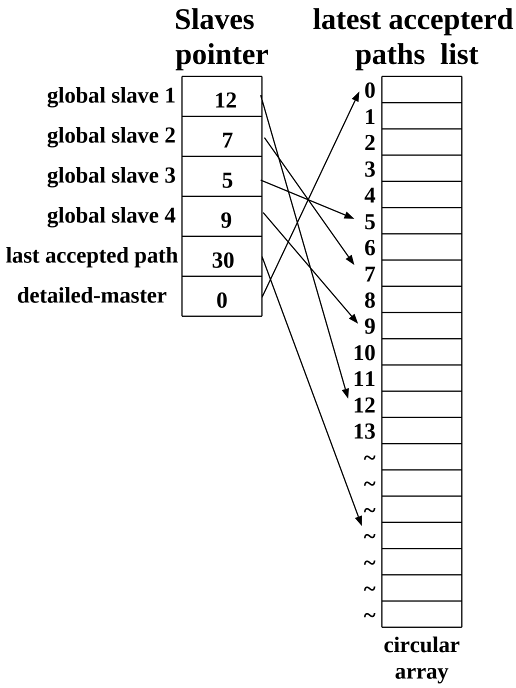


Figure 5.3: Required memory for global-master PE.

in the detailed routing can be occupied by only one net. This is not the case for global routing where the routing is done on the base of the partitions instead of grid points. More than one PE could use the same partition at the same time for routing different nets under the condition that these net paths do not exceed the partition capacity.

CPA does not guarantee to find the least cost path for a net even when using maze running algorithm because the partition costs may be changed by other PEs during calculating this path. This side effect could be decreased if the global-master tries to assign nets from different places to global-slaves at the same time.

The global router has to detect in which layers and in which partitions within these layers each net will be routed. We assume that the number of layers is already known before routing. An example of two layers grid is shown in Figure 5.4a. Each layer is divided into 36 partitions. The shaded area in Figure 5.4b indicates the output of the global router for the two terminals net (S and D). Two layers are assigned to

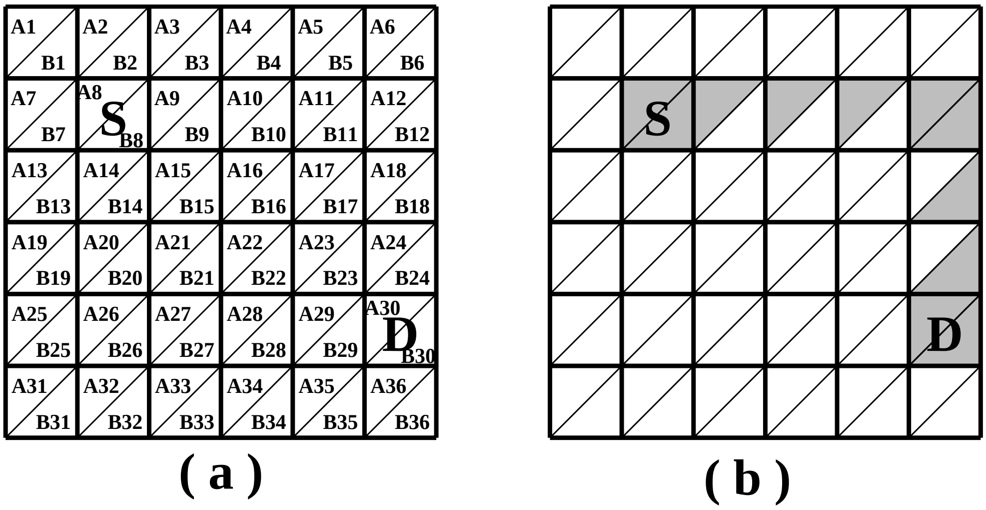


Figure 5.4: Global routing for a net

the partitions which contain net terminals (A8,B8,A30,B30) or path bends (A12,B12). Only one layer is assigned to the other partitions (A9,A10,A11,B18,B24).

We assign two layers to the partitions which contain net terminals to be able to route nets like the example shown in Figure 5.5. In this example, it is required to route both nets A and B (Figure 5.5a). If there is only one layer available for all partitions and we start by routing net A, then net B can not be routed (Figure 5.5b). And if we start by net B, then net A can not be routed (Figure 5.5c). But if there are two layers available at the partitions which contain net terminals, both nets A and B can be routed (Figure 5.5d).

We assign only one layer for the other partitions to increase parallelism among nets during detailed routing. In the detailed routing, nets can be routed simultaneously if their global paths have no common partitions. Figure 5.6 shows an example of 7 nets in which their detailed routing can be done simultaneously.

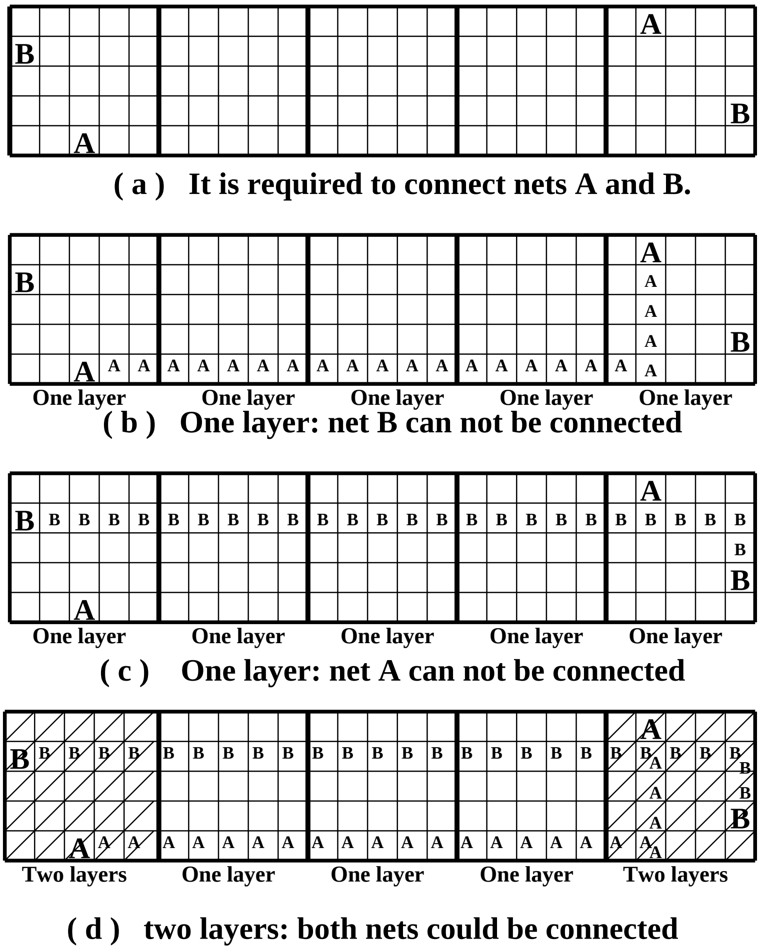


Figure 5.5: Conflicting nets.

5.3 Detailed routing

Although global routing has a restricted X and Y direction, detailed routing allows both X and Y directions on all layers. Without this policy some nets can not be connected. As the global router assigns only one layer at the middle part of the path, so two directions must be available on all layers during detailed routing to be able to route nets like which shown in Figure 5.7. In any layer, the direction for the global routing will be the main (or preferred) routing direction for the detailed routing.

In the detailed routing, the weighted maze running algorithm with its modifications

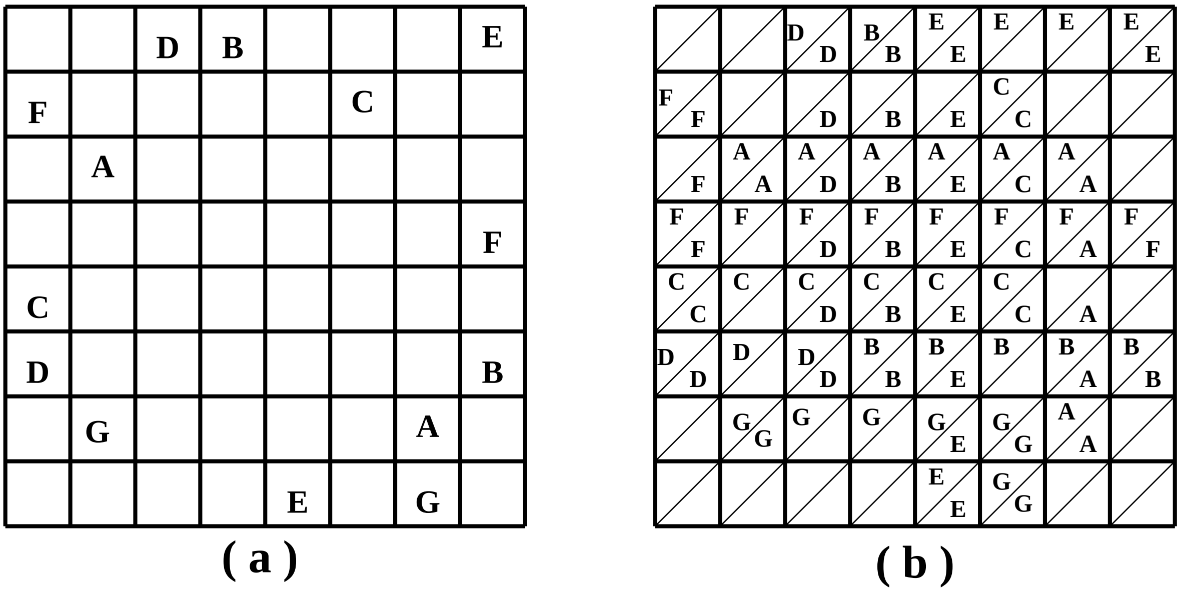


Figure 5.6: Route more than one net simultaneously

is used to find the path for each net. The weight (cost) of each grid point is mainly depend on the direction. For each layer, The cost of propagation in the main (prefered) direction is less than the other direction. i.e. the cost of X direction on odd layers is less than the cost of Y direction on the same layer, and vice-versa for even layers. The cost of vias is much higher.

In the detailed routing we implement both net parallelizm and space parallelizm

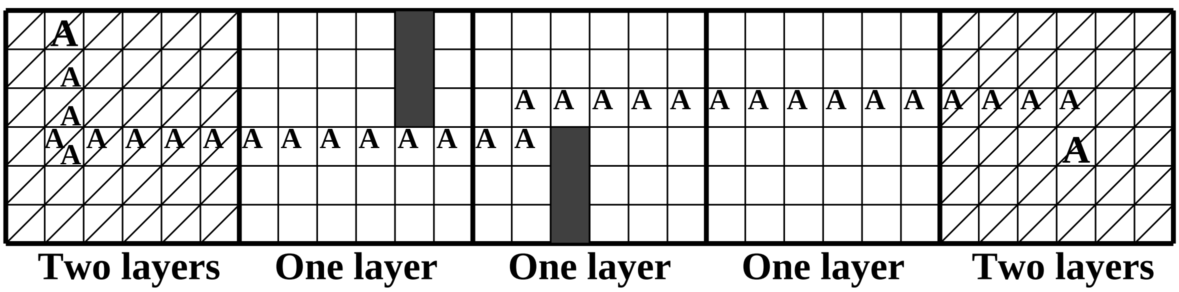


Figure 5.7: Unrestricted direction for detailed routing.

together. The whole routing area is divided among the detailed-slaves. The detailed-master choose the nets in which their global paths has no intersections and sends these nets to the detailed-slaves. The detailed-slaves route their assigned nets within their allocated area.

5.3.1 Conventional previous works on parallel routing

The efficiency of parallelism depends on how to divide the routing space into partitions, and how these partitions are assigned to the PEs. In most cases, the routing space is divided into partitions with equal areas, and every PE will be responsible for one partition. For example, if we have 2 layers grid with dimensions 1000×1000 and there are 16 PEs, then each partition will be $2 \times 250 \times 250$ as shown in Figure 5.8. To route a net shown in Figure 5.9a, global routing detects firstly in which partitions this net will pass (Figure 5.9b).

For parallel detailed routing, some researches [15] [40] locates virtual terminals on the boundaries of the partitions as shown in Figure 5.9c. Then each PE can route the part of the path located inside its partition independently of the other PEs (Figure 5.9d). This way is very fast and all PEs which share routing one net can work simultaneously without any dependence or communication between them. But, the main disadvantage of this method is as follows: As it is very difficult to find the ideal virtual terminals(Figure 5.9e), sometimes undesired paths (long with many vias and bends) will be obtained due to the wrong choosing of virtual terminals (Figure 5.9f).

Some other researches route the net in a sequential manner (one partition after the other). The PE, which is responsible for the partition which contains the source, starts propagation until it reaches the boundaries of its partition(Figure 5.10a). Then, this PE sends the costs of these boundaries to its neighborhoods which continue propagation until also reach the boundaries of the next partitions(Figure 5.10b). These steps are repeated (Figures 5.10c and 5.10d) until the propagation reaches to the destination

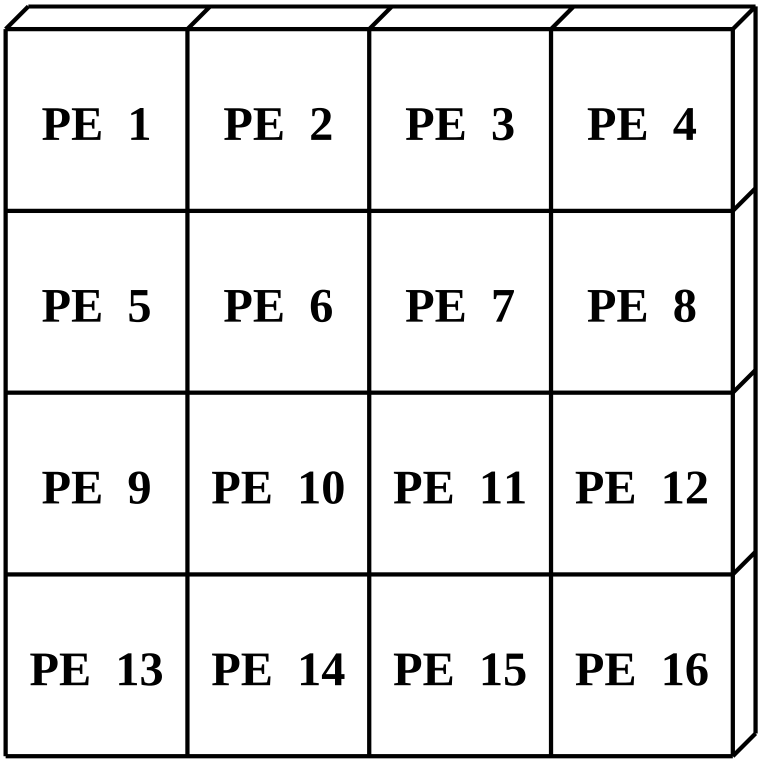


Figure 5.8: Divide the whole routing space into square partitions.

point (Figure 5.10e). This method can obtain a good net path but it uses many messages per net and the parallelizm is poor. The number of messages is proportional with net length. To improve the parallelism, the propagation can start from all net terminals simultaneously. Also, more than one nets can be routed in the same time (Figure 5.10f) if these net paths do not share in any partition (global paths has no intersection).

5.3.2 Slice partitioning

This section explains how to divide the routing space into partitions and how to assign these partitions to the detailed PEs. Detailed PEs are assigned to the layers (dividing the multi-layer grid into layers). If we take the example of having 2 layers grid with dimensions 1000×1000 and 8 PEs for detailed routing, then 4 PEs are assigned to each

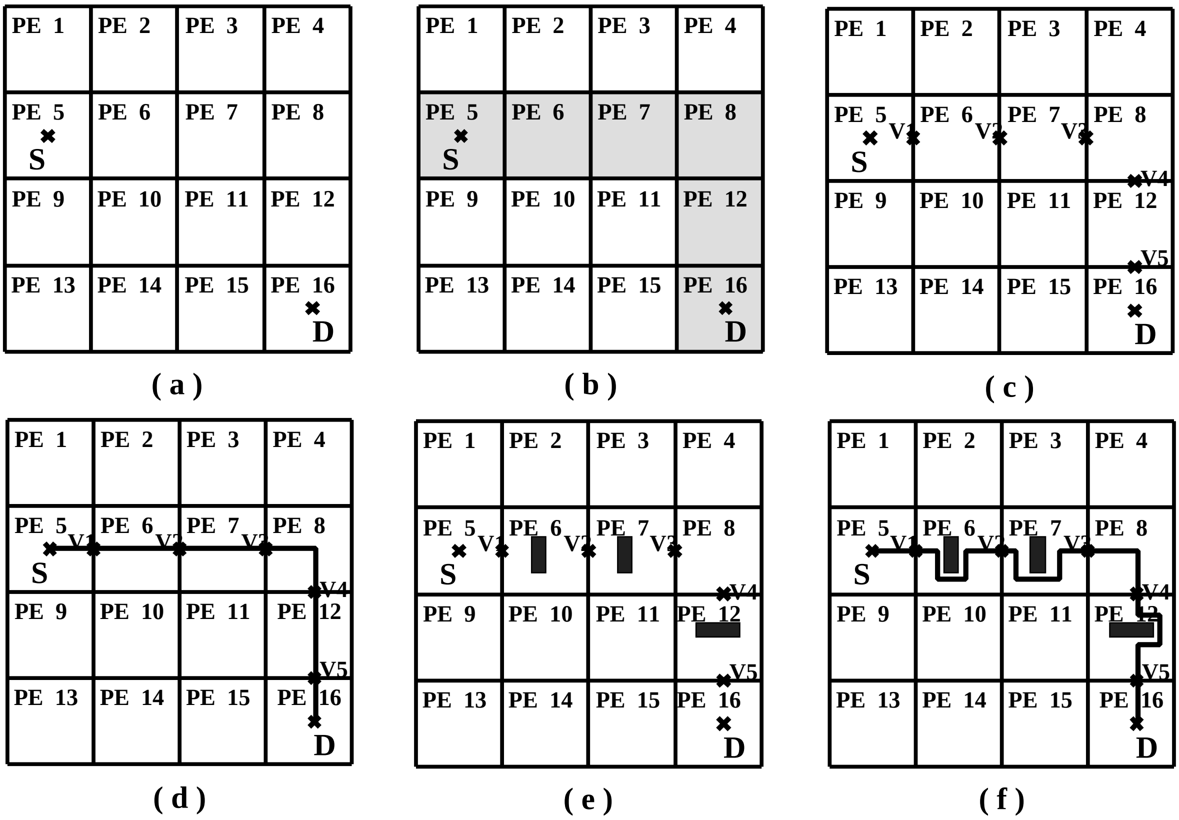


Figure 5.9: Virtual terminals.

layer. Each layer is divided into slices instead of square areas (see Figure 5.11). Each slice length is equal to the grid dimension (1000 in this example), and each slice width is equal to grid dimension divided by $(K \times C)$, where K is the number of detailed PEs assigned to this layer (4 in this example) and C is the number of slices for one detailed PE (2 in this example). As the nets are not uniformly distributed, some parts from the grid are more condensed by nets than the others. Better load balance could be obtained by assigning more than one slice (from different areas) to each PE.

Odd (even) layers are divided into slices in X (Y) direction. Each slice is divided by virtual boundaries into partitions. The virtual boundaries are used to divide the memory of each PE into different parts. Virtual boundaries for odd layers are at the same positions as actual boundaries for even layers and vise versa. By using virtual

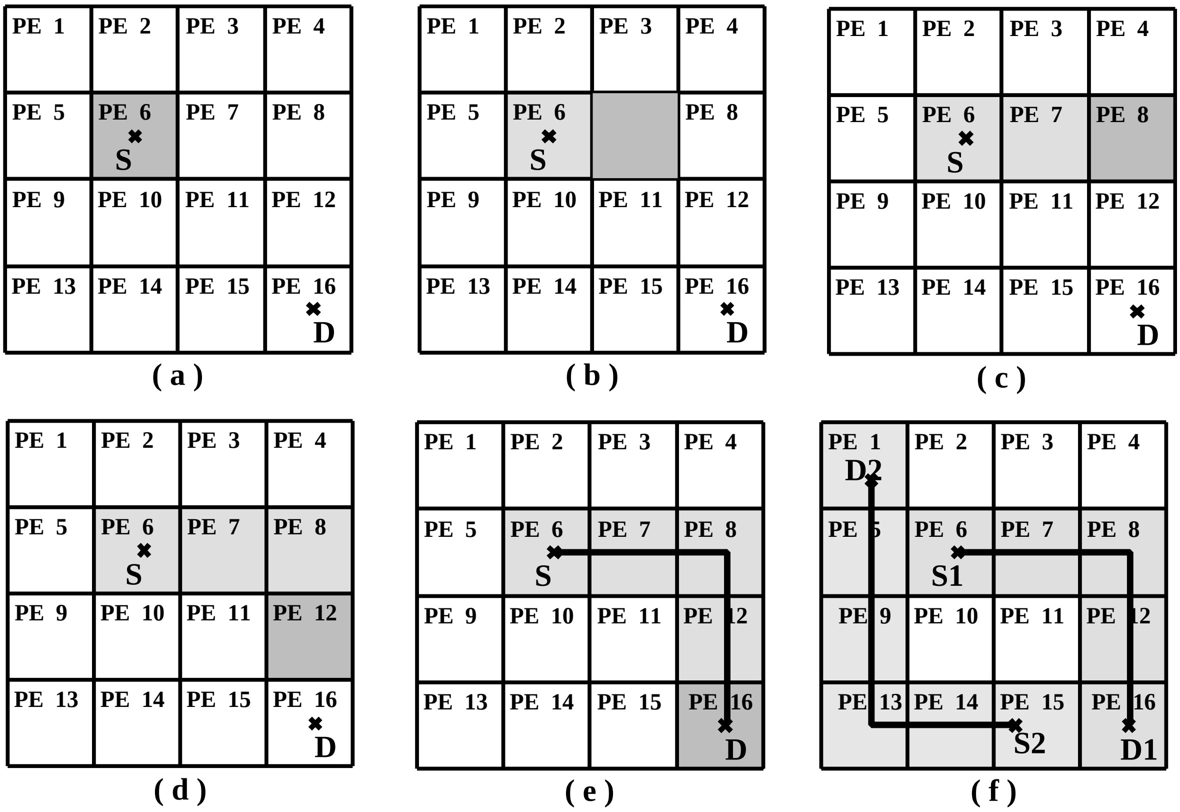


Figure 5.10: Sequential partitions routing.

boundaries, we can have propagation for more than one net in the same slice at the same time if their global paths do not overlap in any partition.

Increasing $K \times C$ decreases the slice width. This decreases the propagation area for detailed routing, and thus speeds up the detailed routing processing. On the other hand, this makes the global routing more difficult (number of partitions becomes larger), and the successful connection ratio may be decreased (if slices become too narrow).

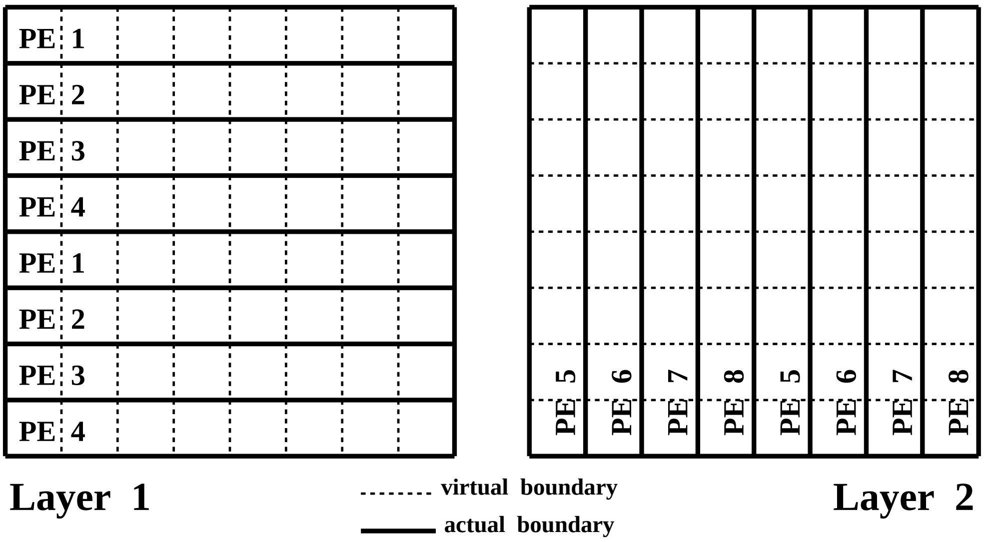


Figure 5.11: Detailed grid assignment

5.3.3 Algorithm of detailed routing

This section explains how the detailed routing is done in parallel and how each detailed PE communicates with the others to route the nets. This is done in a master-slave manner. We exploit parallelism in both nets and space. The master PE checks if there is a net which has all the partitions on its global path marked free, sends this net identification to the detailed PEs, marks the partitions on its global path as busy, and then deletes this net from the group. The detailed PEs, which have the partitions located on this global path, work cooperatively together to find the detailed path for this net. Each PE finds the part of the detailed path which lies within its slice and sends a message to the master PE to free the partitions used for routing this net. These steps will be repeated for all nets.

Figure 5.12 shows an example of the detailed routing for one net. If the global path of this net is as shown in Figure 5.12a, the detailed routing is done as in the following steps.

1. If all the partitions on the global path for this net are marked free, then the

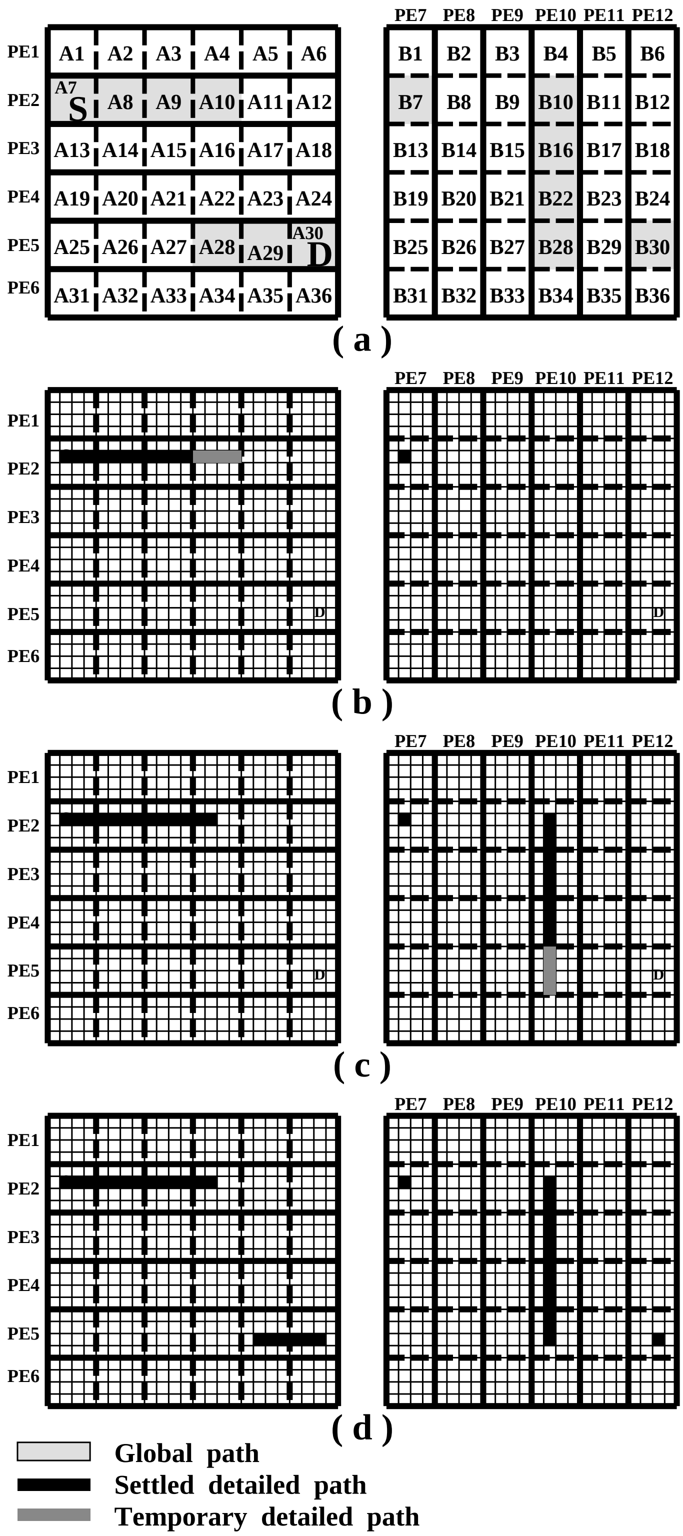


Figure 5.12: Detailed routing for a net.

master marks these partitions as busy, sends this net to PE_7.

2. PE_7 sends the bit map of partition B7 to PE_2. This bit map informs PE_2 of occupied grid points (by any other nets or obstacles). PE_7 now can not use B7 but it can use any other partitions within its slice to route other nets.

3. PE_2 finds a detailed path (using maze running algorithm) between the source and the end of the global path within its slice (virtual boundary between A10 and A11). This detailed path consists of two parts: settled (A7, A8, A9) and temporary (A10) (Figure 5.12b).

4. PE_2 sends bit map of B7 back to PE_7, so PE_7 can now use this partition again in routing other nets. PE_2 sends message to the master PE to free partitions (B7, A7, A8, A9), sends the bit map of partition A10 to PE_{10}. Now, PE_2 can not use partition A10 only for routing other nets.

5. PE_{10} finds a detailed path between the received temporary detailed path in A10 and the end of net global path within its slice (Figure 5.12c).

6. PE_{10} sends bit map of A10 back to PE_2, sends message to the master to free partitions (A10, B10, B16, B22), sends bit map of partition B28 to PE_5, sends message to PE_{12} to send bit map of B30 to PE_5.

7. PE_5 finds a detailed path between the destination and the received temporary detailed path (Figure 5.12d), sends bit map of B28 back to PE_{10} and B30 back to PE_{12}, sends a message to the master to free partitions (B28, A28, A29, A30, B30).

As the master checks whether all partitions on the global path of any net are free before sending this net to the detailed PEs, dead locks are never happen. This also guarantees that only one net can be routed in any partition in the same time. The partitions which contain temporary paths could not be used to route other nets until this temporary path will be freed or settled. This is because the part of this temporary path (which will be freed later) may prevent another net to be routed, so this net will be marked as it could not be routed though it may be routed after this part be freed. Although this may decrease the parallelizm but it decreases the number of failed routed nets.

The main advantages of this technique are:

1. Decrease of the communication cost since the number of messages per net is proportional with number of vias instead of net length.

2. Improve resulting routing paths by using less number of vias.

3. Improve parallelism by allowing large number of nets to be routed simultaneously.

4. obtain good paths for the nets (no virtual terminals).

5.4 Results

Amon has been implemented on AP1000 parallel computer and tested using 3 industrial MCM (multi-chip module) benchmarks [1] shown in Table 5.1.

MCC1-75 (75 micron grid) consists of 6 chips, 765 I/O pins and contains 799 signal nets, two power and one ground net. There are 2496 pins total, 2043 of which are signal pins. There are numerous 3 to 7 pin nets. The power and ground nets are not routed on the x and y layers, but are tied directly to the power and ground planes. There are two chip footprints: 550 $\times$ 550 mils with 448 pins, and 330 $\times$ 330 mils with 272 pins

MCC2-75 (75 micron grid) and MCC2-45 (45 micron grid) are a models of the next generation supercomputer on a 6 x 6 inch substrate with 37 Honeywell VHSIC gate arrays and 18 high density connectors. The chips are 1.5 x 1.5 cm. The net list contains 7118 signal nets and 14659 pins.

[1]These data are available via anonymous ftp from mcnc.org

Table 5.1: MCM benchmark

	chips no.	nets no.	pins no.	grid size
MCC1-75	6	802	2495	599 $\times$ 599
MCC2-75	37	7118	14659	2032 $\times$ 2032
MCC2-45	37	7118	14659	3386 $\times$ 3386

Table 5.2 shows a comparison between the results of Amon and some sequential algorithms [41] [42] run on Sun SparcStation II. As reported in [42], sequential 3D maze failed to produce a routing solution for MCC2-75 and MCC2-45 because of its high memory requirement. Sequential slice router [41] also failed to produce a routing solution for MCC2-45.

The routing ratios are reported as 100% for these sequential algorithms. Both 3D maze and Slice use one more layer than our router. Amon obtained connection ratio of 97.5%, 98%, and 99% for MCC1-75, MCC2-75, MCC2-45 respectively. The successful routing ratio of V4R only is better than Amon. Amon obtains 100% connection ratio for all these three benchmarks when using one more layer (the same layer number as sequential 3D maze or sequential slice). Amon uses fewer vias and shorter wire length than all the other three sequential algorithms. This is mean that Amon could obtain better routing quality.

Table 5.3 shows the speedup obtained by Amon using 32 PEs when comparing with different sequential algorithms running on Sun SparcStation II. Amon runs 272 times faster than sequential maze for MCC1-75. For MCC2-75, Amon obtains 154 and 21 times speedup comparing with sequential slice and V4R respectively. The numbers of slices per layer are 30, 80 and 120 for MCC1-75, MCC2-75 and MCC2-45 respectively.

Table 5.4 shows total routing time (global + detail) using different number of PEs. In our program, we use at least one PE for each layer plus master and global PEs. Also, as we use maze algorithm, the total required memory for the whole routing space

Table 5.2: Comparison with some sequential algorithms on Sun SparcStation II

	Program	MCC1-75	MCC2-75	MCC2-45
Number of layers used	seq. 3D maze	5	-	-
	seq. slice	5	7	-
	seq. V4R	4	6	4
	Amon	4	6	4
Unrouted nets	Amon	20 (2.5%)	142 (2%)	70 (1%)
Time (sec)	seq. 3D maze	3540	-	-
	seq. slice	720	29700	-
	seq. V4R	180	3960	5820
	Amon (32 PEs)	13	193	324
Total wire length	lower bound	343767	5362181	8935372
	seq. 3D maze	397221	-	-
	seq. slice	402258	5902818	-
	seq. V4R	394272	5559479	9130705
	Amon (32 PEs)	382479	5498146	9051632
Number of vias	seq. 3D maze	8794	-	-
	seq. slice	6386	47864	-
	seq. V4R	6993	36438	36473
	Amon (32 PEs)	2825	22823	19554

Table 5.3: Speedup of Amon (32 PEs) comparing with sequential algorithms

Program	MCC1-75	MCC2-75	MCC2-45
seq. 3D maze	272	-	-
seq. slice	55	154	-
seq. V4R	14	21	18

for MCC2-75 or MCC2-45 is much more than the available memory per one PE. For these two reasons, we could not calculate the speed up comparing with one PE, and we have to compare it with some minimum number m of PEs which is 6, 10, and 6 for MCC1-75, MCC2-75, and MCC2-45 respectively. We define parallelizability for n processors as

$$\text{parallelizability} = m \times T_m / T_n$$

where

T_m: time using the minimum number (m) of PEs.

T_n: time using n PEs.

Figure 5.13 shows the time against number of processors for MCC1-75, MCC2-75, and MCC3-45 in a logarithmic scale. Figure 5.14 shows the parallelizability for different number of PEs. As could be seen from these two figures, Amon has a good scalability as the time is not saturated yet even though we use up to 300 PEs for both MCC2-75 and MCC2-45.

Table 5.4: Parallelization result

	no. of PES					time	time	paralleliz-
	global		detail		total	(sec)	× PEs	ability
	M	S	M	S				
MCC1-75	0	1	1	4	6	62	372	6
	1	2	1	8	12	32	384	11.63
	1	2	1	12	15	25	375	14.88
	1	2	1	20	24	16	384	23.25
	1	3	1	24	29	14	406	26.57
	1	4	1	40	46	11	506	33.82
	1	6	1	60	68	9	612	41.33
	1	13	1	120	135	7	945	53.14
MCC2-75	1	2	1	6	10	592	5920	10
	1	5	1	12	19	316	6004	18.73
	1	11	1	24	37	172	6364	34.42
	1	14	1	30	46	150	6900	39.47
	1	23	1	48	73	95	6935	62.32
	1	28	1	60	90	85	7650	69.65
	1	48	1	120	170	56	9520	105.7
	1	78	1	240	320	40	12800	148
MCC2-45	0	1	1	4	6	1645	9870	6
	1	2	1	8	12	830	9960	11.89
	1	3	1	12	17	628	10676	15.72
	1	4	1	16	22	474	10428	20.82
	1	5	1	20	27	383	10341	25.77
	1	7	1	24	33	324	10692	30.46
	1	9	1	32	43	250	10750	39.48
	1	11	1	40	53	210	11130	47
	1	13	1	48	63	179	11277	55.14
	1	17	1	60	79	145	11455	68.07
	1	22	1	80	104	123	12792	80.24
	1	33	1	120	155	95	14725	103.9
	1	43	1	160	205	82	16810	120.4
	1	58	1	240	300	67	20100	147.3

M:master , S:slave

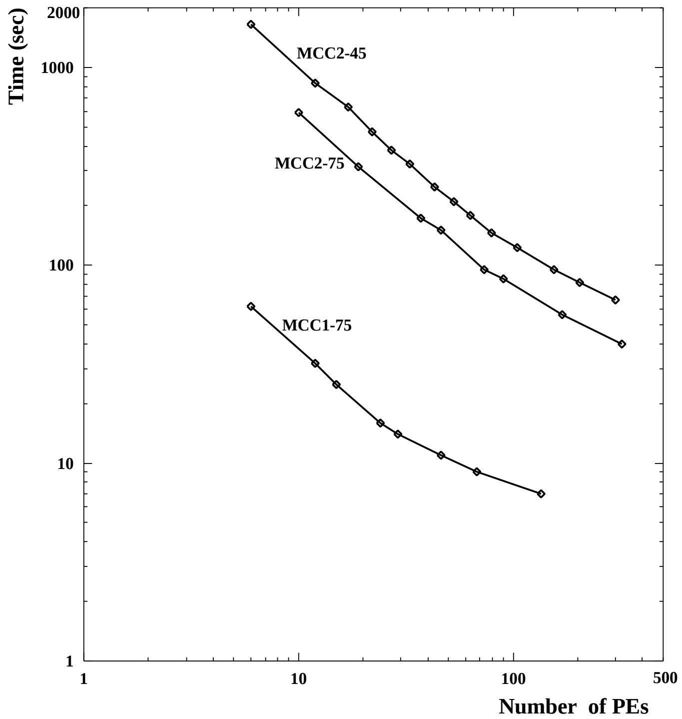


Figure 5.13: Time against number of PEs in logarithmic scale.

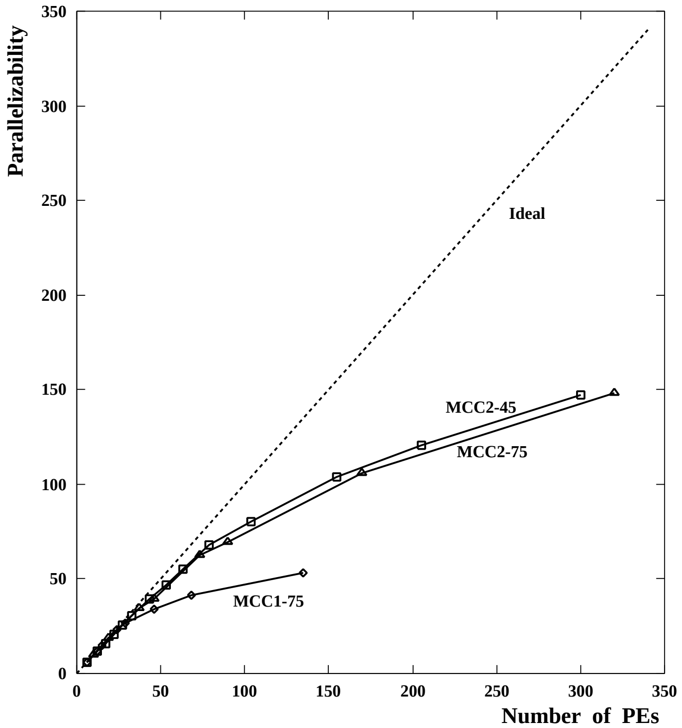


Figure 5.14: Scalability.

Chapter 6

Modifications of Amon

In this chapter, we introduce two new modifications of Amon to improve the successful routing ratio to be as the best sequential algorithm. The first modification introduces new cooperative relation between global and detailed routers. This cooperative relation allows the global PEs to get the actual number of available tracks (*capacity*) inside the partitions instead of estimating these capacities. The second modification is a new algorithm for detailed routing called maximum capacity algorithm (*max-cap*). Max-cap algorithm finds the maximum capacities for the slices, which are on the global path of the net, and routes each net within these slices by a path which decreases each capacity by only one. The results show that using max-cap algorithm in the detailed routing obtains higher connection ratio than using maze running algorithm.

In addition to improving the routing ratio, we try to improve the speed and to decrease the communication cost by introducing two phases algorithm (Amon2). In the first phase, we use the rotating area algorithm introduced in Chapter 4 to route short nets using only few messages. In the second phase, we use the slice based grid partition algorithm introduced in Chapter 5 to route the rest of the nets. An improvement in the speed is obtained by this merge without affecting the routing ratio.

6.1 Introduction

In the previous chapter a parallel slice maze router, called Amon, has been introduced. Amon has high degree of parallelism and good speed up comparing with the fastest sequential algorithm. On the other hand, Successful routing ratio for Amon is less than that for V4R (a sequential routing algorithm) [42]. In this chapter, we introduce two new techniques to improve the successful routing ratio of Amon. By using these techniques, the routing ratio becomes almost the same like the best sequential routing algorithm. Unfortunately these two techniques slightly decrease the parallelism and increase the computation time.

The first technique introduces a new cooperative relation between global and detailed routers. This cooperative relation allows the global PEs to get the actual number of available tracks (*capacity*) inside the partitions instead of estimating these capacities. In many routers [13] [15], global routers have to estimate the capacity inside each partition because these routers separate completely global from detailed routing processes(like what we have done in the previous chapter). Although it is very hard to get complete accurate estimations, the accuracy of these estimations have great effects on the routing ratio. In this chapter, instead of estimating the capacities, the detailed router calculates the exact capacities of all partitions, and feeds these capacities back to the global router. The results show that to feed-back the capacity improves the successful routing ratio at the expense of longer computation time and less parallelization.

The second technique is a new algorithm for detailed routing called maximum capacity algorithm (*max-cap*). Maze running algorithm has a serious disadvantage especially for detailed routing that previous routed net paths may prevent other nets from being routed later. Max-cap algorithm tries to overcome this disadvantage. Max-cap algorithm finds the maximum capacities for the slices, which are on the global path of the net, and routes each net within these slices by a path which decreases each capacity by only one. Both max-cap and maze running algorithms are implemented on a parallel computer using MCM (Multiple Chip Module) benchmarks. The results show that using max-cap algorithm in the detailed routing obtains higher connection ratio than using maze running algorithm. On the other hand, max-cap algorithm is slightly slower than maze running algorithm.

In addition to improving the routing ratio, we try to improve the speed and to decrease the communication cost. For this purpose, we combine both Amon and the rotating area algorithm into a two phases algorithm. In the first phase, we use the first phase of the rotating area algorithm introduced in Chapter 4 to route short nets using only few messages. In the second phase, we use the slice based grid partition algorithm introduced in Chapter 5 to route the rest of the nets. An improvement in the speed is obtained by this merge without affecting the routing ratio.

6.2 Cooperation between global and detailed routing

In the previous chapter, the flow of data was in one direction only from global PEs to detailed PEs without any feedback of data from detailed PEs to global PEs. The nets are divided into groups. The global PEs find the global paths for the nets within each group, then send these paths to the detailed PEs. The detailed PEs find the detailed path for each net within the partitions located on the global path for this net. The global PEs have no information about these detailed paths and there is no way for the global PEs to know the exact partition capacities. The global PEs has to estimate these partition capacities.

One method for this estimation is to initialize the boundary capacity as equal to the length of the boundary between two partitions, and decrease this capacity by one when a global path of any net crosses this boundary. This method is simple, but in many cases it could not give the accurate capacity. In this section, we will introduce a more accurate method to calculate the partition capacity.

6.2.1 Aspects to be considered while calculating a partition capacity

We define the partition capacity as the maximum number of net paths that could be drawn between the left edge (left boundary) and the right edge if the main routing direction of this partition is in X direction, or between bottom and top edges if the main routing direction is in Y direction. We will consider only the case in which the main routing direction is the X direction because the other case will be almost the same.

To get the accurate capacity of a partition, the locations of all obstacles and the paths of the previous routed nets within this partition must be known. Moreover, the obstacles in the adjacent partitions must also be known because the obstacles in one partition affects the capacities of the adjacent partitions. The following examples are given to illustrate the different aspects which must be considered while calculating the partition capacity.

Considering the boundary capacity only is not enough. The capacity within the partition itself must be considered. Figure 6.1a shows an example of a partition which contains 8 terminals for 8 different nets. Although 4 nets have to cross each of the boundaries in the X direction in all cases shown in Figure 6.1b,c and d, only 4 nets from 8 could be routed in Figure 6.1b, all the 8 nets could be routed in Figures 6.1c and 6.1d. There is also a space which can be used to route another 2 nets in Figure 6.1d as indicated by the dashed lines.

Considering only the number of obstacles inside the partition is not enough to get the partition capacity, locations of these obstacles must also be considered. For example, the capacities for the partitions shown in Figure 6.2 are different, though each of these partitions has 6 obstacles. For example, the pins of the ICs decrease the capacity in one routing direction more than the other direction.

Routing a net may or may not decrease the capacity of the partitions which contain the terminals of this net. This is depends on how the detailed path of this net is drawn.

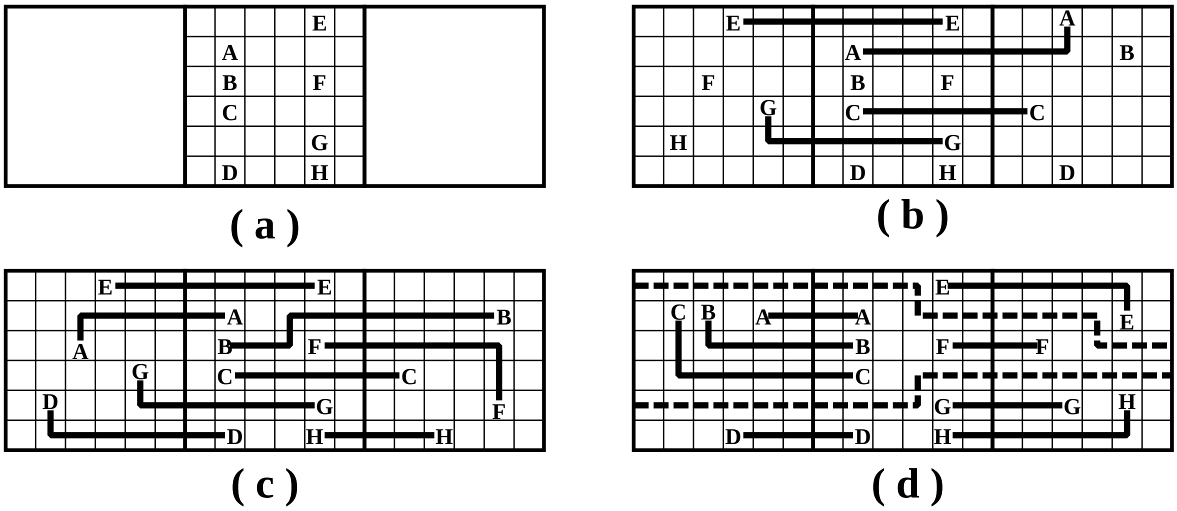


Figure 6.1: Boundary capacity only is not enough.

For example, the capacity of the partition shown in Figure 6.3a will be decreased by 1 if net A is connected as shown in Figure 6.3b, but it will not be decreased if net A is connected as shown in Figure 6.3c. For this reason, updating the partition capacity for global routing must depends on how the detailed paths are drawn.

The obstacles in one partition may affect the adjacent partitions. We will call this effect as mutual effect between adjacent partitions. This mutual effect must be considered while calculating the overall cost for a path. In some cases, although all capacities of some adjacent partitions are greater than zero, no detailed path could be drawn between these partitions. In Figure 6.4a, there is no path between T1 and T2 though each of the partitions has a capacity equal to 3. Neglecting the mutual effect between adjacent partition will lead the global router to choose a path which could not be drawn by the detailed router. We overcome this problem by making overlapping between partitions while calculating the capacities as shown in Figure 6.4b. In this Figure, the overlapping step is equal to half of the partition length. By using this overlapping, the capacity of the second partition from the left becomes zero (see Figure 6.4b) which mean that there is no path between net terminals T1 and T2.

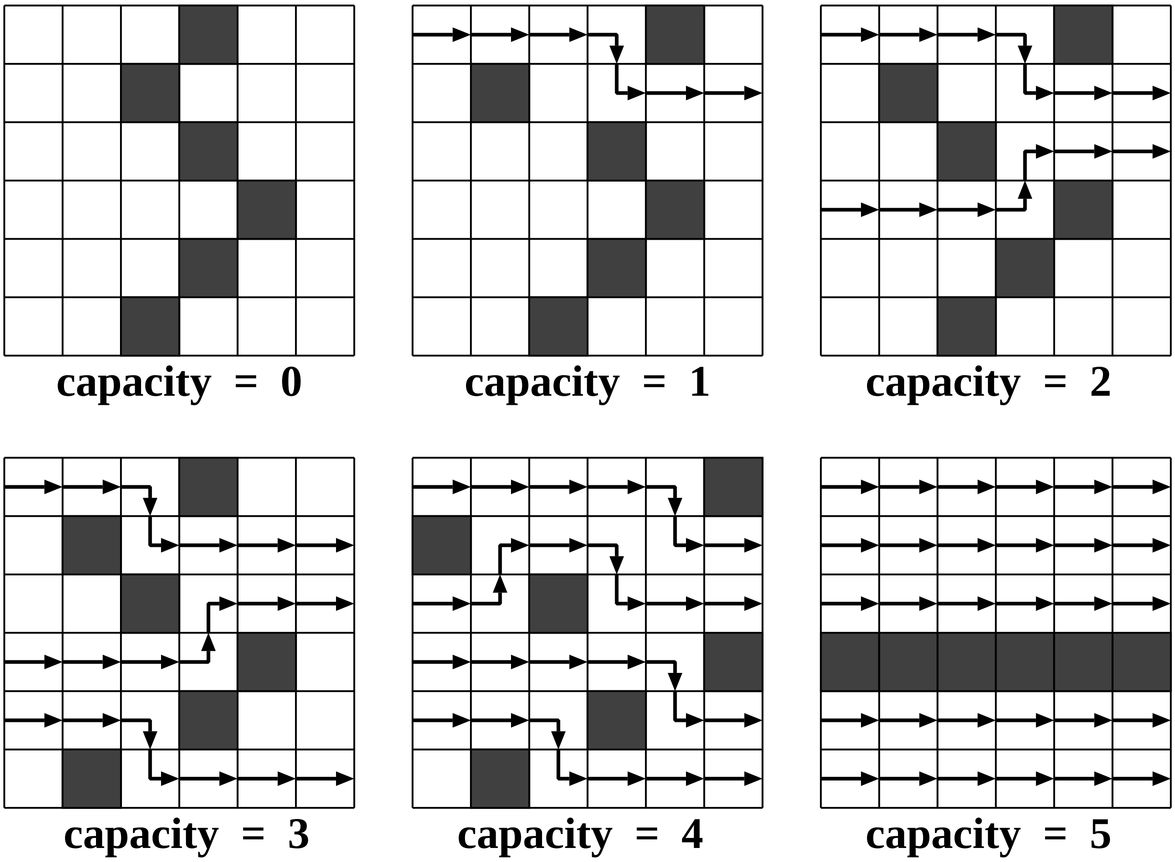


Figure 6.2: Capacity depends on locations of net terminals.

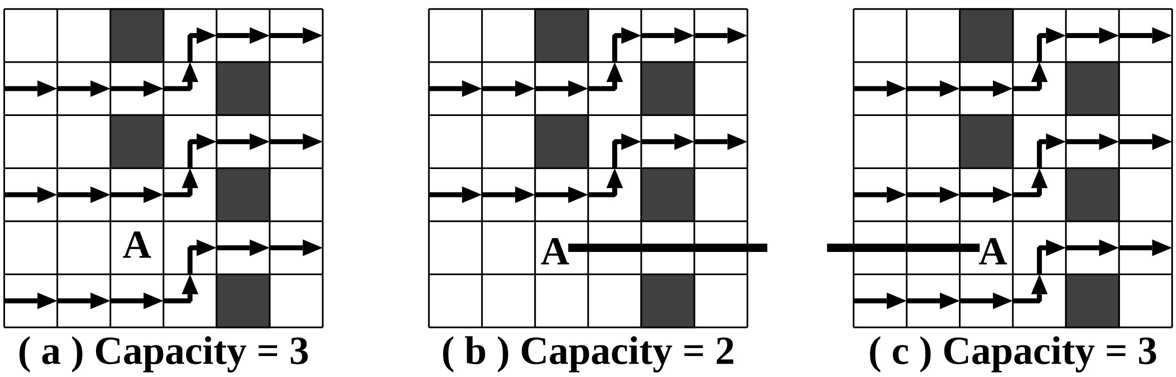


Figure 6.3: Updating capacity depends on detailed paths.

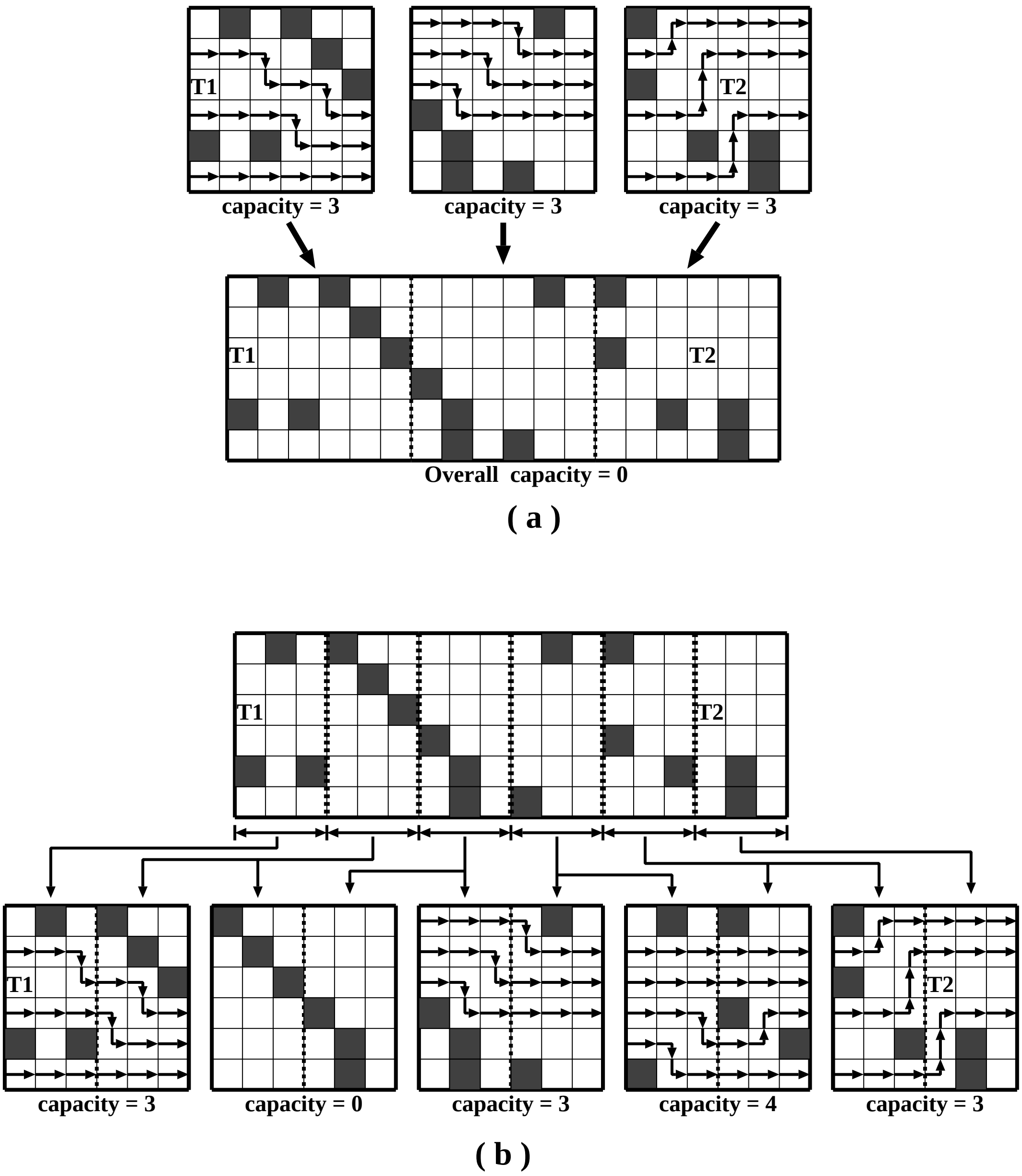


Figure 6.4: Overlapping between partitions.

6.2.2 Finding the actual partition capacity

Now, we will introduce a simple algorithm to find the actual capacity for any partition. At the beginning all the grid points are labeled as either free or occupied. Occupied means that these points are representing obstacles, net terminals, or paths of the previous routed nets. Only free points could be used for routing new nets. To find the actual capacity of a partition, we draw net paths one after the other from bottom to top until there is no more net paths to be drawn. Each path is drawn through free points starting from the left edge and ending at the right edge.

To get the partition capacity of the example shown in Figure 6.5a, the algorithm checks the label of the points located on the left edge, starting from bottom to up. If any point from these points is free, then this point will be considered as it is visited from left and marked by $L(\leftarrow)$ as shown in Figure 6.5b. The location (address) of this point will be stored in a pointer. We will call the grid point which has its address stored in this pointer as the active point. The algorithm checks the adjacent neighbors grid points to this active point in a certain order. This order, as shown in Figure 6.5c and Table 6.1, depends on the direction in which the presently active point has been visited before. For example, if the active point is marked by $L(\leftarrow)$ which mean that it has been visited from left, then the algorithm checks the adjacent neighbors points in this order: 1:down, 2:right, 3:up. If any point of these adjacent points is free, then the algorithm stops to check the other points, stores the address of this point in the pointer (this point will be the active point instead of the previous point), labels this point by the direction of which it has been visited from. In the example shown in Figure 6.5d the new active point is the right adjacent point to the previous active point which means that it has been visited from left.

Until arriving at the right edge as shown in Figure 6.5e, the algorithm continues to do the following steps: 1: checks the adjacent points to the active point in the order depending on the direction in which the presently active point is visited as shown in Figure 6.5c, 2:considers the first free point from these adjacent points as the new active

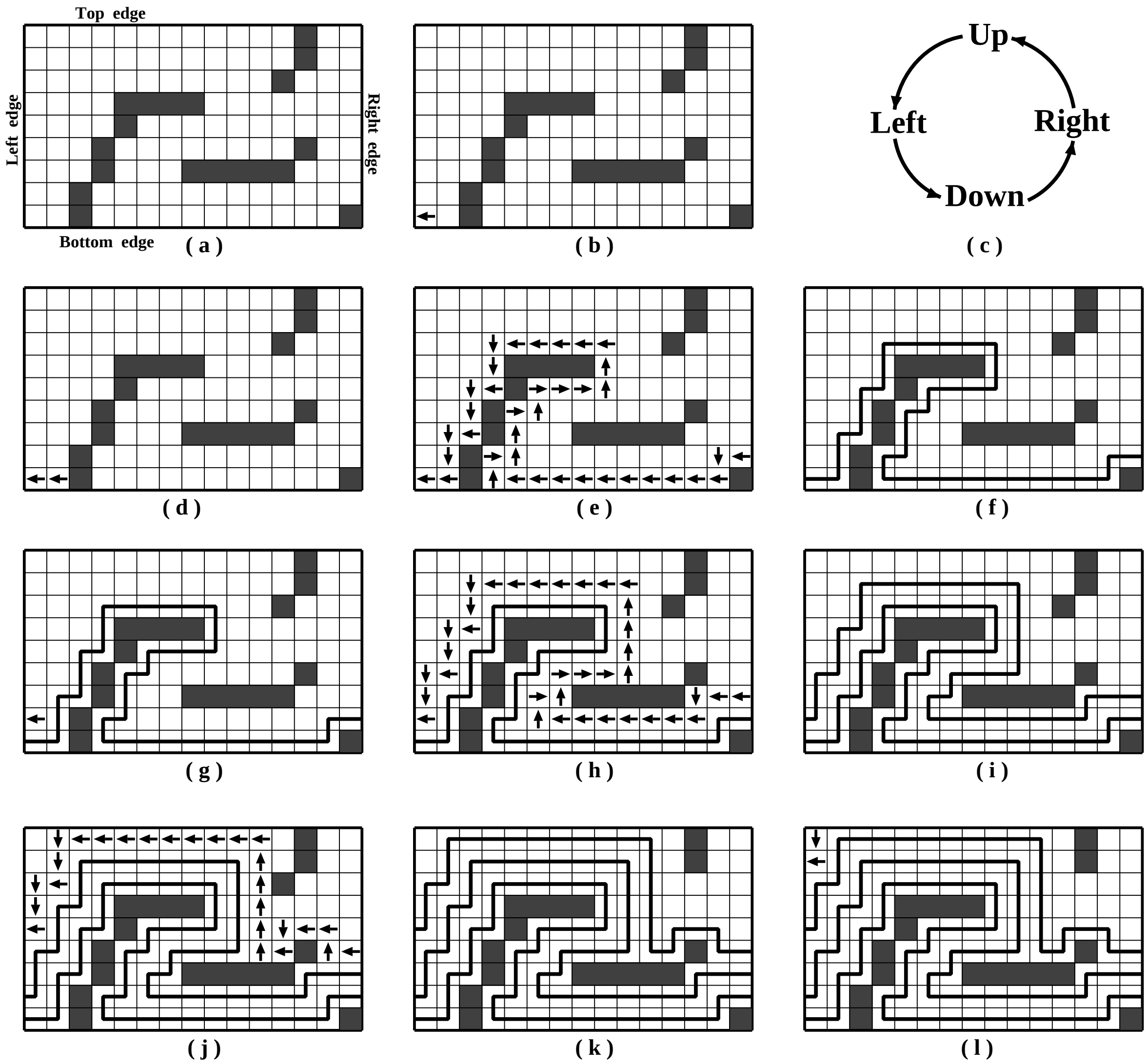


Figure 6.5: Finding the actual capacity of a partition.

Table 6.1: Order of checking adjacent points to the active point.

active point	Order to check the adjacent points		
direction	check first	check second	check third
Left	Down	Right	Up
Down	Right	Up	Left
Right	Up	Left	Down
Up	Left	Down	Right

point, and 3: labels this new active point by the direction in which it has been visited from. Arriving at the right edge means that one net path could be drawn as shown in Figure 6.5f. If all the adjacent points to any of the active points are not free, then the algorithm return back to the previous active point using the direction stored in the presently active point.

To find another net path, the algorithm searches again for the first free point on the left edge from bottom (Figure 6.5g), checks the adjacent points until it arrives at the right edge (Figure 6.5h). Then the second path could be drawn as shown in Figure 6.5i. These steps will be repeated until no more paths could be drawn as shown in Figures 6.5j, 6.5k and 6.5i.

Three bits in the memory be needed to store the status of each grid point. The grid point may take one of 6 cases (*000:free* , *111:occupied* , *010:Left* , *011:Right* , *100:Up* , *101:Down*). The flowchart to get the partition capacity is shown in Figure 6.6.

6.2.3 Feedback the actual capacity to the global PEs

In Amon, the PEs are divided into global PEs which perform global routing and detailed PEs which perform detailed routing. The whole routing area is divided among the detailed PEs to decrease the required memory. The actual capacity of a partition depends on the detailed paths of previously routed nets and locations of obstacles and

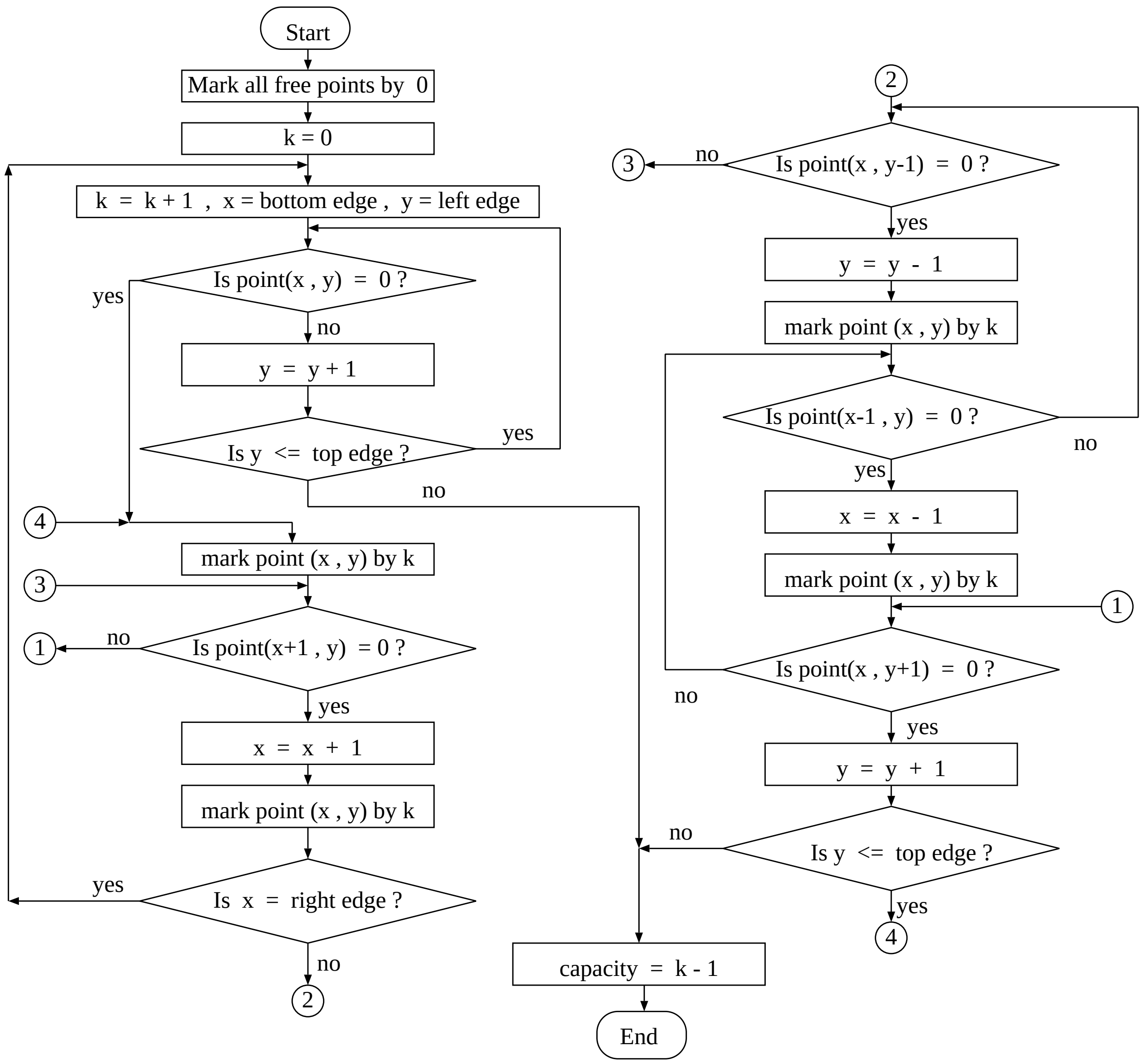


Figure 6.6: Flowchart of finding the actual capacity.

net terminals inside this partition. Each detailed PE is the only PE which knows the detailed data of its assigned partitions. From this, we can see that we have the following situation: although the actual capacity will be used by global PEs, the required data to calculate these actual capacities is owned by detailed PEs. There are two solutions to deal with this situation.

The first one is to send all the data of the partitions from detailed PEs to global PEs. Then, global PEs will use these data in calculating the actual capacities. We do not use this solution because the amount of data required to represent all the partitions is too large. This solution will cause a huge amount of communication and extra memory will be required for global PEs.

The second solution is to calculate the actual capacities using detailed PEs. Each partition capacity is represented by only one integer number. Each detailed PE calculates the actual capacities of all of its assigned partitions, and then, sends a message which consists of few integer numbers to the global-master PE. We use the second solution because it causes less communication cost than the first one, and it does not need extra memory for global PEs.

If the detailed PEs calculate the actual capacities after routing each net and feed them back to the global PEs, the global PEs will always have the accurate capacities. On the other hand, both the computation time and the communication cost will be dramatically increased. To compromise between getting accurate capacity and computation time, many nets are collected together to form one group, and the actual capacities will be calculated only after routing all the nets within each group. When the global PEs route one group, the global paths of this group will be sent to the detailed PEs. The detailed PEs find the detailed paths for this group, and then calculate the actual capacities of all partitions and send these capacities back to the global PEs to update their capacities by actual data. The global PEs then start to route another group and so on until all the groups will be routed as shown in Figure 6.7a.

This means that only at the beginning of routing each group, the global router will update its capacities by the accurate (actual) values. Within the same group, after

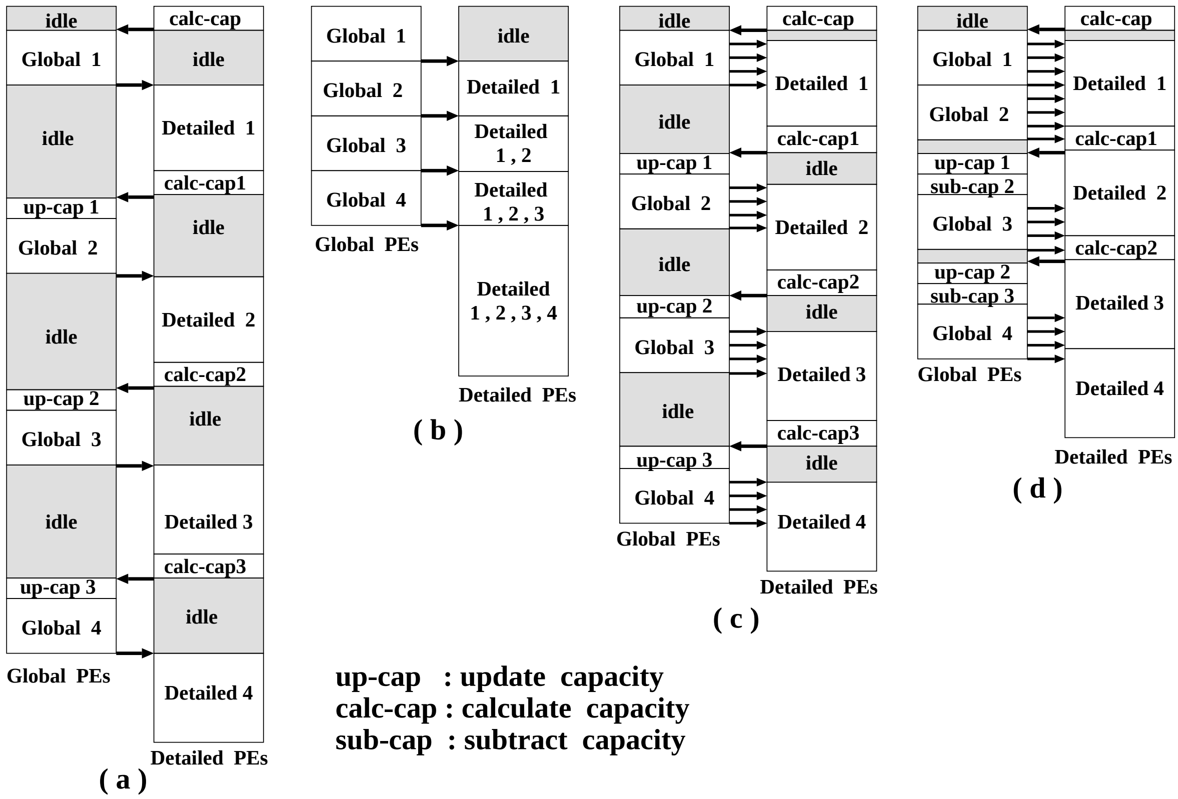


Figure 6.7: Alternating between global and detailed routing.

finding the global path of a net, the global router will subtract one from the capacities of the partitions located on this global net path.

Figure 6.7b shows the case of the previous chapter in which there is no feedback from detailed PEs to global PEs. When the detailed-master PE receives global paths for any group, it will add these paths to the other nets paths which still unrouted from previous received groups. There is no need for the detailed-master PE to separate nets from different groups. Dividing nets into groups in this case does not decrease the parallelization among nets in the detailed routing because detailed-master PE does not separate nets from different groups.

Comparing Figures 6.7a and 6.7b shows that feedback increases the idle time for both global and detailed PEs. In fact the idle time for detailed PEs is much important than for global PEs for two reasons: 1: Number of detailed PEs is larger than the number of global PEs. 2: The required time for detailed routing is usually longer than for global routing. This mean that the global PEs will finish global routing for all nets, then they have to wait until the detailed PEs finish detailed routing.

In Figure 6.7a, detailed PEs are waiting until receiving the global paths of any group from the global PEs. To decrease this idle time, each group is divided into several sub-groups. The global PEs send the global paths of each sub-group to the detailed PEs to start detailed routing earlier as shown in Figure 6.7c. Increasing number of sub-groups decreases the idle time but it increases the number of messages. Un-equal sub-groups strategy could also be used in which the first sub-group will be small, then the second sub-group will be larger than the first, the third is larger than the second, and so on.

The idle time could be further decreased if global PEs start routing for group number m after receiving the actual capacity of group number $m-2$ (before receiving the actual capacity of group number $m-1$) as shown in Figure 6.7d. Global PEs subtract one from the capacities (actual capacities of group $m-2$) of the partitions which located on the global paths of group number $m-1$. By this earlier starting, there is no idle time for detailed PEs (except at the beginning during the global routing of the first sub-group). It is better not to increase this earlier starting for global PEs by more than one group. This is because the earlier starting for global routing may have some bad effect on the accuracy of the capacities used for global routing.

In case of using feed-back, dividing nets into groups decreases the parallelization among nets. The detailed-master PE has to separate between nets from different groups. At the ends of each group, some detailed PEs will be idle as there is no enough jobs for all PEs. Moreover, if most of the nets within any group are concentrated in some partitions only, then the detailed PEs which are responsible about these partitions will be heavily loaded while the other detailed PEs will be idle. This problem is known as load balance problem. Now, we will discuss three methods to overcome this problem:

1. Static load balance: The first solution is to consider the locations of nets while dividing them into groups. Each group must have nets which are distributed on the whole routing area. The global-master chooses the nets in each group

depending only on the locations of net terminals because this choosing is done before finding the global paths. Locations of net terminals only could not give an accurate equal distribution because two nets with different terminals locations may and may not share the same global partitions. The main advantage of this method is that it does not increase the computation time of the whole routing process. This is because it does not increase communication cost and the extra computation required by global-master is not so big.

2. Dynamic load balance: When the detailed-master received the global path of one group, it calculate the number of nets in each slice and re-assigns the slices to the detailed PEs. This solution can be easily done in case of shared memory computer. But if the computer is message passing, as in our case, this solution will increase the communication cost as the whole data of the slices must be transfered between different PEs. Also extra memory space may be required as some detailed PEs may have more slices than the other detailed PEs.

3. If one detailed-slave is idle, then this idle detailed-slave will take some slices from another detailed slave which has a high load. This solution increases both the computation time and communication cost, but it could obtain better load balance.

We use only the first method (static load balance) which provides some improvements in load balance without any extra communication cost or computation overhead. The other two methods will be considered in the future work.

6.3 Max-capacity algorithm for detailed routing

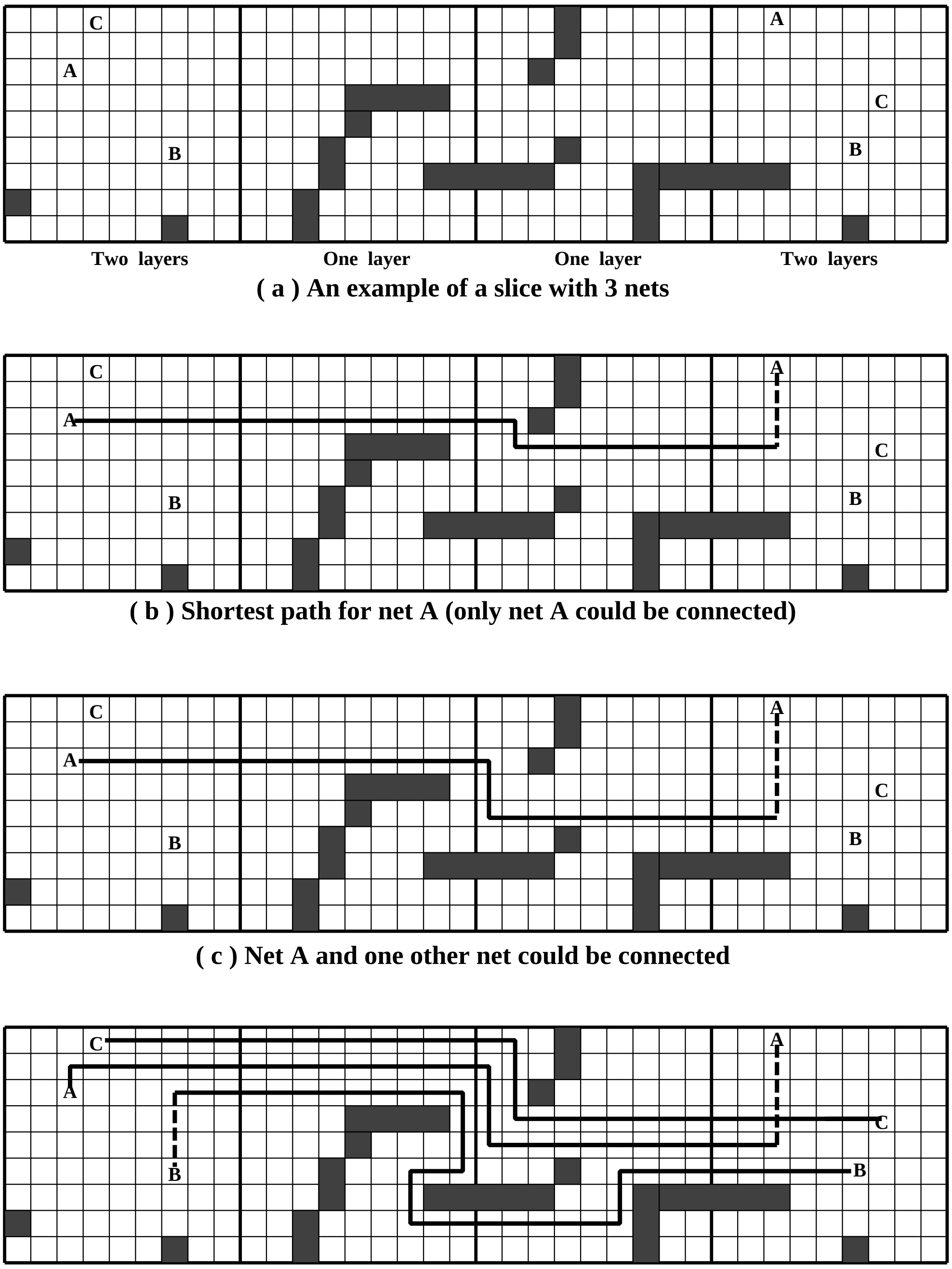
Although maze running algorithm has serious disadvantage specially for detailed routing that previous routed net paths may prevent the other nets from being routed later, it still gives higher connection ratio than many other algorithms. Rip-up and reroute have been introduced to overcome this disadvantage, but it needs long computation time. It is also difficult to decide which nets have to be rerouted.

This disadvantage becomes more severe in our detailed router as we assign only one layer for some partitions. Figure 6.8a shows an example of 3 nets (A, B, and C). As there is only one layer available in the middle partitions, the shortest path for net A in this example will prevent the other nets from being routed later as shown in Figure 6.8b. Only one more net (B or C) can be routed if net A is routed as shown in Figure 6.8c. All the three nets can be routed if we choose the optimal paths shown in Figure 6.8d.

Now, we will introduce a new algorithm, called *max-cap*, for detailed routing which can obtain these optimal paths. Max-cap finds the maximum capacities for the slices, which are on the global path, and routes each net within these slices by a path which decreases each capacity by only one.

The max-cap algorithm routes each net under the condition that this net path will not decrease any partition capacity by more than one. To route a net using this algorithm, the global path for this net is divided into slices. Each slice consists of the partitions located on the global path within the same layer. For example, the global path for the net shown in Figure 6.9 consists of two slices. The first slice contains the partitions A8, A9, A10, A11 and A12, and the second slice contains the partitions B12, B18, B24 and B30. For each slice, we find the maximum number of paths (capacity) which can be routed between the two edges of the slice as explained before. Then connect the net by the nearest path from these paths.

Figure 6.10 shows an example to explain how to get the slice capacity. In this example, 3 paths could be drawn (Figure 6.10f). These paths have a lot of unnecessary bends and long length. To improve the quality of these paths, delete the last drawn path (first one from the top), find the shortest path which connects the left edge to the right edge and draw it (Figure 6.11a,b). Then, delete the next path, find the shortest one (Figure 6.11c,d), and so on. These operations will be repeated for all paths (Figure 6.11). Note that we can also stop these steps when arriving at any



(a) An example of a slice with 3 nets

(b) Shortest path for net A (only net A could be connected)

(c) Net A and one other net could be connected

(d) All the 3 nets could be connected

Figure 6.8: Previous routed nets may prevent other nets from being routed later in maze running algorithm.

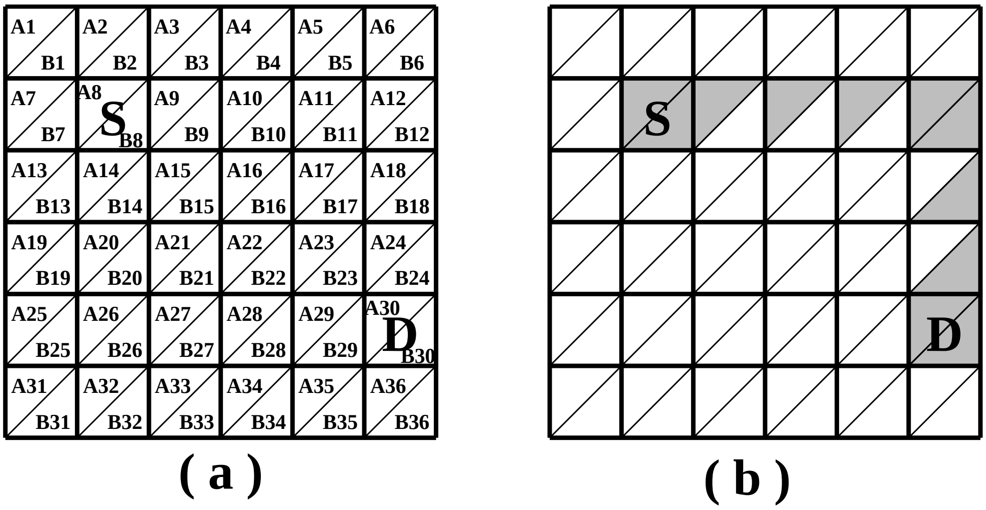


Figure 6.9: Global routing for a net

terminal of net A which we want to route. At the end, we choose the nearest path from these paths to connect net A (Figure 6.11g). This path for net A does not blocked the other nets (B, C).

6.4 Amon2: a two phases algorithm

As short nets take short time for routing, the ratio between communication and computation for a message passing computer becomes worst even if only one message is used per net. To overcome this problem, we introduce a new router which consists of two phases with two different grid assignment. Each phase consists of cooperative global and detailed routers. In both phases, we use the max-cap algorithm for detailed routing. The first phase is aimed to route all short nets using few messages, then the rest of the nets will be routed in the second phase.

In the first phase, we use the idea of rotating the assignment area for every PE as in the rotating area algorithm introduced in Chapter 4. We modified the grid division of this algorithm to improve the load balance. In the second phase, we use a slice based grid assignment introduced in Chapter 5. Table 6.2 shows the main features of the two

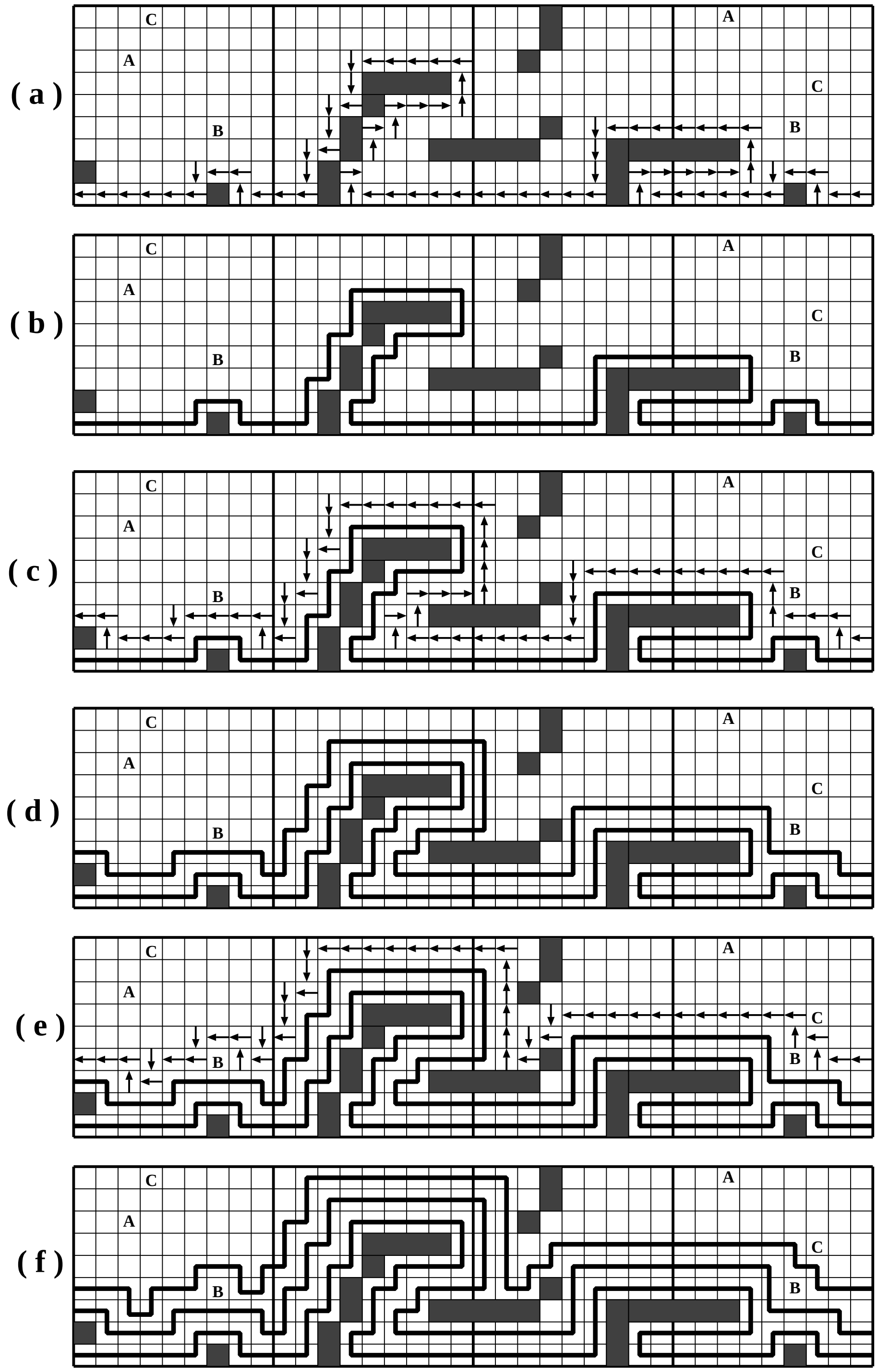


Figure 6.10: Max-cap algorithm (find the maximum slice capacity).

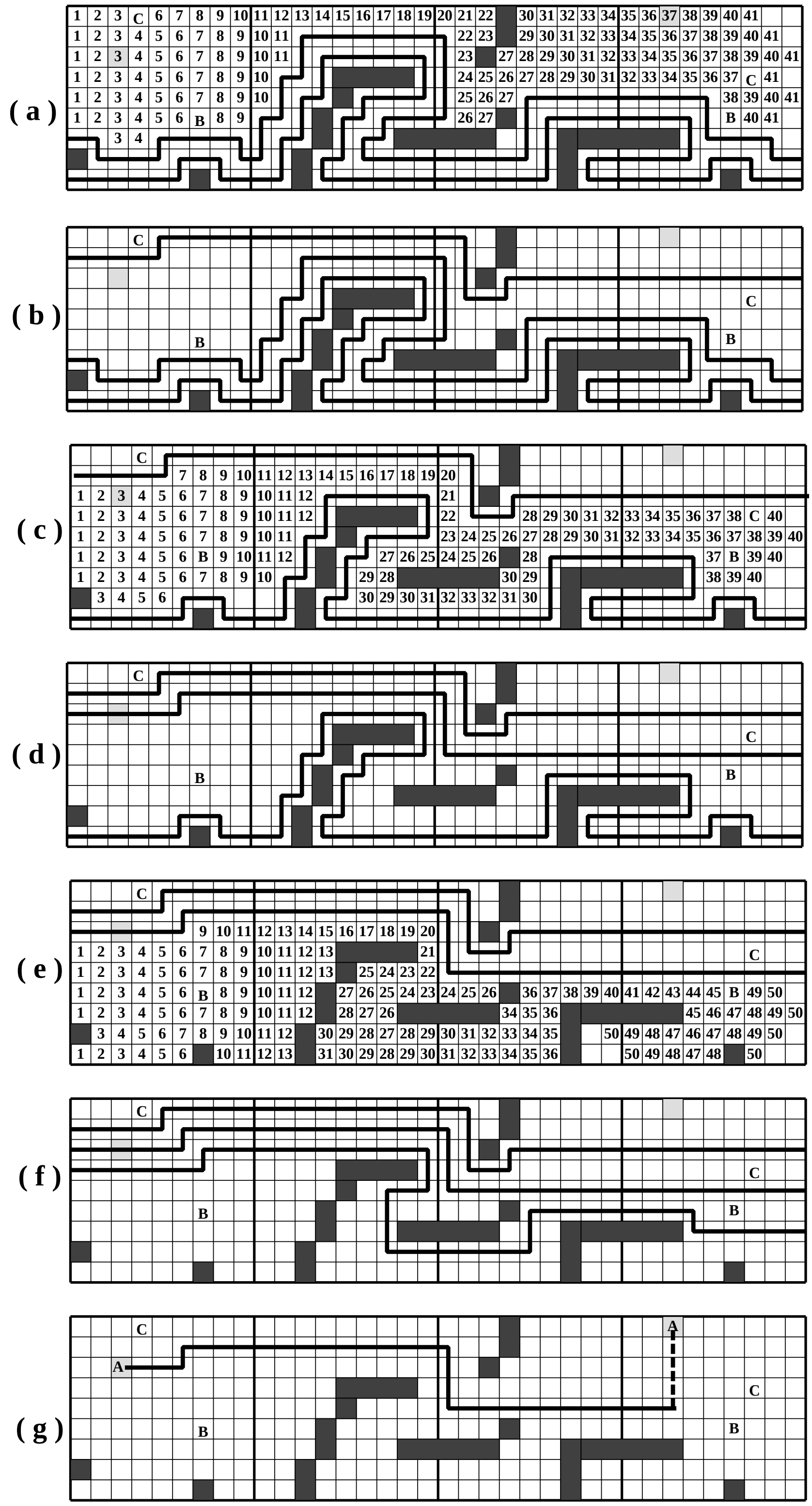


Figure 6.11: Max-cap algorithm (improve the paths quality).

Table 6.2: Two phases algorithm

	First phase	Second phase
Objective nets	Short nets	Long nets
PEs processing	For each partition, both global and detailed routing executed on the same PE.	PEs are divided into global PEs and detailed PEs.
Global algorithm	Maze running algorithm with different costs.	Maze running algorithm with different costs.
Detailed algorithm	Max-capacity algorithm.	Max-capacity algorithm.
Cooperation	Cooperative global and detailed routing within the same PE.	Cooperative global and detailed routing between global PEs and detailed PEs.
Parallel algorithm	Rotating area algorithm.	Global: Competing processors algorithm. Detailed: Slice based grid assignment.
Parallelism	Global: Net parallelism. Detailed: Net parallelism.	Global: Net parallelism. Detailed: Net and space parallelism.

phases.

6.4.1 First phase (for short nets)

As short nets take short time for routing, then the communication overhead should be minimized. In this phase, we rotate the area assignment to each PE to route all short nets using only few messages as we have done in the first phase of the rotating area algorithm introduced in Chapter 4. We extent the rotating area algorithm for multi-layer grid instead of single layer, divide the routing process into global and detailed routing processes to improve the speed, use max-cap instead of maze running algorithm for detailed routing to improve the successful routing ratio, and modify the grid assignment to improve load balance as will be explained in this section. In this phase, global routing is used to decrease the computation time required for detailed routing.

The routing process is divided into global and detailed processes with cooperative relation between them. Every PE executes both global and detailed routing processes on all the nets within its assigned area. Every PE selects the nets which lie in its assigned area, sorts them in an ascending order according to their lengths, divides them into groups, and routes all the groups (both global and detailed) one after the other. The real capacities will be calculated only after finishing routing one group. Within the same group, the capacities are updated by subtracting one from the partitions on the global path. We do not calculate the real capacity after routing each net to limit the computation time required for this calculation. As both global and detailed routing are done on the same PE, cooperation between global and detailed does not need any communication cost. Moreover, dividing nets into groups does not decrease the parallelization in this phase.

In this phase, the grid is divided among all PEs as shown in Figure 6.12a. There is a part of the routing area which is not assigned to any PE as indicated by the shaded areas in Figure 6.12. Consider that the width of this part is equal to S, the number of PEs is N^2, and the grid dimensions are $L \times L \times M$, where M is the number of layers. Every PE is responsible for an area of $d^2 \times M$, where $d = (L - S)/N$.

If we choose $S = L/(2N+1)$ as in Chapter 4, then d will be equal to $2S$, and all PEs will get equal areas during all the four steps as originally introduced in Chapter 4. The problem of load unbalance may happen even if the nets are equally distributed. For example, in the case of equally distributed grid, PEs in the last column in Figure 6.12b (PE8, PE16, PE24, ... ,PE64) have much load than the other PEs since there is a part of their assigned grid area which has not been visited before. So the other PEs have to wait for these 8 PEs till finish their work. This will be occured again for the PEs in the last row in Figure 6.12c and for PE57 in Figure 6.12d.

This drawback could be overcome by decreasing the value of S. Then d, which is equal to $(L - S)/N$, will be larger than $2S$. In the first step, the assigned area to all PEs will be equal to $d^2 \times M$. In the second step, the area assigned to the PEs in the last column will be $(0.5d + S) \times d \times M$ which is smaller than the area assigned to the

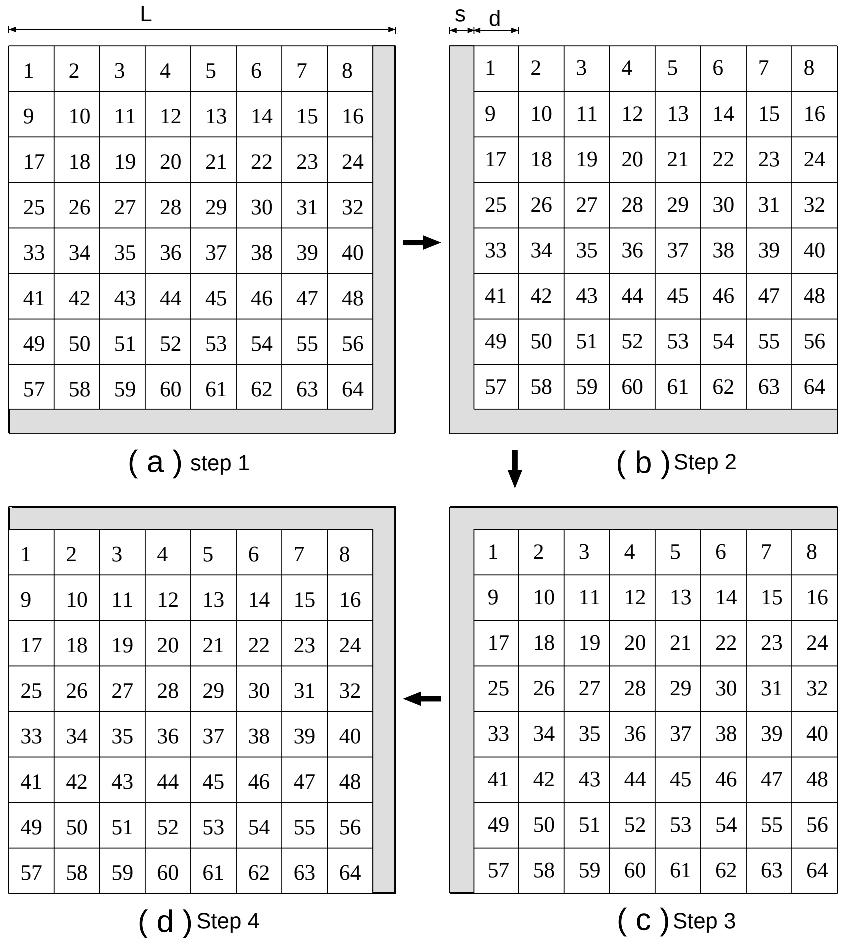


Figure 6.12: Dividing the routing area among PEs in the first phase.

other PEs (which still equal to $d^2 \times M$). This difference in areas improves load balance because it decreases the load for the PEs in the last column to be equal to the load of the other PEs. In the third step, the area assigned to the PEs in the last row will also be smaller than the area assigned to the other PEs.

This phase may again suffer from the load unbalance in case of unequal distributed grid (short nets are concentrated in the areas assigned to few PEs). To overcome this problem, the host will stop this phase and start the second phase if the number of active PEs becomes less than certain ratio from the total number of PEs (50% for example). Unequally grid partition strategy [40] may also be used to improve the load balance,

```
For all PEs in parallel
    Mark all nets by 0
    For all the four directions
        Select the nets located in its assigned area.
        Sort these nets in ascending order according to net length
        Divide these nets into groups.
        For all groups
            Calculate the actual capacity of the partition in the assigned area
            For all nets inside this group and marked by 0
                Find global path
                Decrease capacities on the global path by one
                Find detailed path
                If detailed routing is succeeded for this net
                  then mark this net by 1
            end for all nets
        End for all groups
        Send data to your neighbor in this direction
        Receive data from your neighbor in the opposite direction
    End for all directions
Send paths to the host. End for all PEs.
```

Figure 6.13: The first phase program.

but the memory size required by each PE will be different and communication cost and computation overhead will be increased, so that we do not adopt this approach.

6.4.2 Second phase (for long nets)

We use the slice based grid assignment introduced in Chapter 5. PEs are divided into global PEs and detailed PEs. Max-cap algorithm is used for detailed routing. The cooperation (feedback the actual capacity) between global and detailed routing processes is also used.

6.5 Results

Five programs shown in Table 6.3 have been implemented on AP1000 (MIMD, distributed memory parallel computer with torus network [21]) using C language to examine the effectiveness of the algorithms described in this Chapter. These programs have been tested using 3 industrial MCM (multi-chip module) benchmarks [1] shown in Table 5.1 and 2 test data shown in Table 6.4. In generating the test data1 and data2, we write a program to generate nets with random X and Y terminals locations. The Manhattan distances of the nets obey the Rayleigh probability functions. Data2 has many sets of nets, each set has a different average Manhattan distance. For all these sets, the total length of the nets within the same set(average net distance $\times$ nets number) is constant and equals to 2,000,000.

6.5.1 Successful routing ratio

Maze versus max-cap

Using max-cap algorithm gets higher successful routing ratio than the well known maze running algorithm. The numbers of unrouted nets for MCC1-75, MCC2-75, MCC2-45, and data1 become 3, 35, 7, and 8 respectively when using max-cap algorithm, these

[1] These data are available via anonymous ftp from mcnc.org

Table 6.3: Different programs to determine algorithms effectiveness

	Detailed routing		Capacity		Number of phases	
	Maze	max-cap	Without feedback	With feedback	One phase	Two phases
prog1	○		○		○	
prog2		○	○		○	
prog3		○	○			○
prog4		○		○	○	
prog5		○		○		○

Maze : using maze running algorithm for detailed routing.
max-cap : using max-cap algorithm for detailed routing.
Without F.B : Detailed router does not send the exact capacity to Global router.
With F.B : Detailed router sends the exact capacity to the global router.
One phase : Using only the second phase for both short and long nets.
Two phases : Using first phase to route short nets then use the second phase for long nets.

Table 6.4: Test data

	chips no.	nets no.	pins no.	grid size	Layers	Lower bound of wire length
data1		32000	64000	3024 × 3024	4	9,600,000
data2		variable	variable	1000 × 1000	4	2,000,000

Table 6.5: Time and number of unrouted nets

	Number of PEs			Layers	prog1		prog2		prog3		prog4		prog5	
	G	D	T		time (sec)	unr. nets	time (sec)	unr. nets	time (sec)	unr. nets	time (sec)	unr. nets	time (sec)	unr. nets
MCC1-75	14+1	48+1	64	4	9	20	10	3	10.5	3	13.3	0	13.8	0
MCC2-75	14+1	48+1	64	6	105	142	127	35	131	34	173	9	176	8
MCC2-45	10+1	52+1	64	4	179	70	246	7	251	7	316	0	322	0
data1	10+1	52+1	64	4	193	52	228	8	212	8	321	0	250	0

G: Global D: Detailed T: Total unr. nets: unrouted nets numbers are 20, 142, 70, and 52 respectively when using maze running algorithm. This improvement is done at the expense of execution time as shown in Table 6.5 (compare prog1 and prog2).

Cooperation between global and detailed routing

Much more improvement in the successful routing ratio is obtained by using actual capacity in the global routing. The numbers of unrouted nets when using the actual capacity (prog4, prog5) become 0, 9, 0 and 0 for MCC1-75, MCC2-75, MCC2-45 and data1 respectively

On the other hand, the computation time is increased by the time required for calculating the actual capacities. Moreover, dividing nets into groups decreases the parallelism among nets as only few PEs are simultaneously working at the end of each group as shown in Figure 6.14 (prog4).

6.5.2 Using one phase algorithm versus two phases algorithm

One phase algorithm

In prog2 and prog4, we use only the second phase to route both short and long nets. Short nets are firstly routed as the nets are already sorted in ascending order according to their length. Figure 6.14 shows that the number of PEs which are simultaneously working for routing short nets is fewer than for long nets (few number of PEs are working at the starting of both prog2 and prog4). The communication cost for routing short nets is higher than for routing long nets as could be seen from Figure 6.15.

Two phases algorithm

Using two phases algorithm (prog3 and prog5) increases the number of PEs simultaneously working in routing short nets (Figure 6.14), and decreases the communication cost as only few messages are used in the first phase to route all short nets (Figure 6.15).

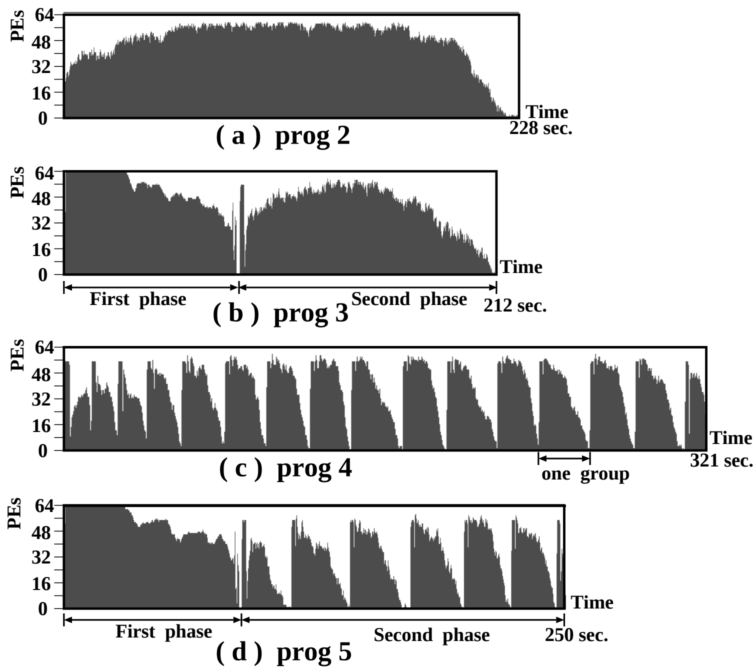


Figure 6.14: Number of PEs simultaneously working in routing data1.

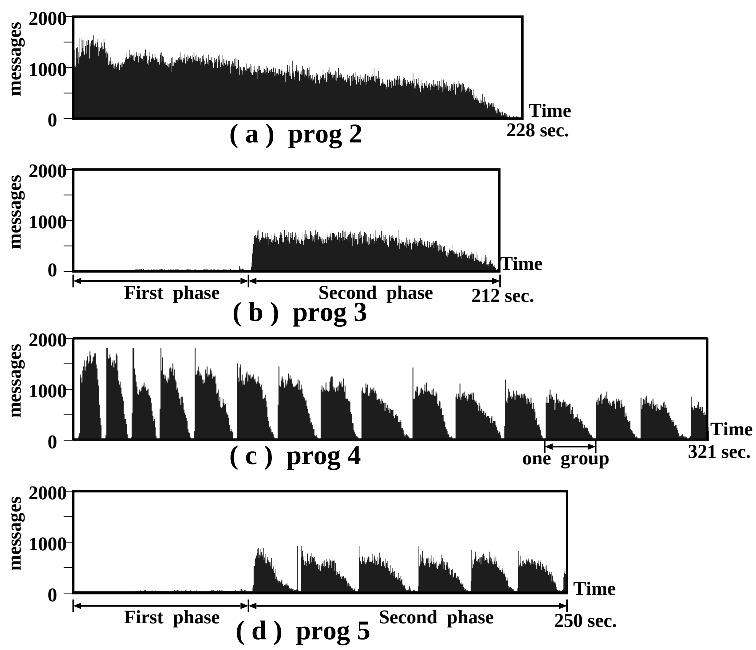


Figure 6.15: Number of messages (communication cost) in routing data1.

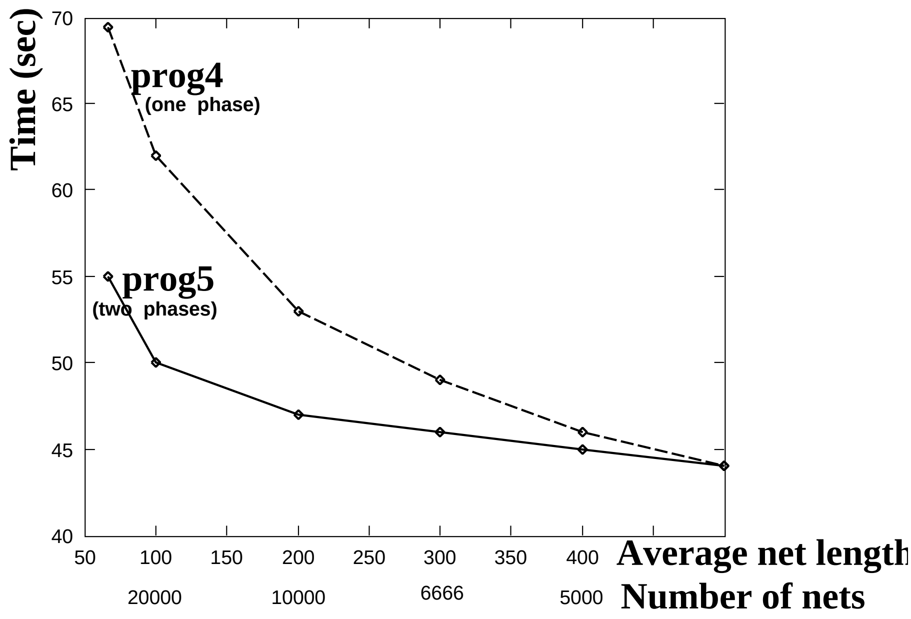


Figure 6.16: Effectiveness of using two phases for different net-lengths.

Dividing nets into groups does not affect the parallelism in the first phase.

Using two phases decreases the execution time without affecting the successful routing ratio. The execution time required to route data1 using only one phase (prog4) is 321 seconds which will be decreased to 250 seconds when using two phases (prog5). As there is only few short nets in these MCM benchmarks, so using the first phase to route short nets does not improve the speed (Table 6.5). Another experiment has been done using data2 to measure the effect of using two phases on different net lengths. Figure 6.16 shows time against net length for both prog4 and prog5 using 64 PEs. Using two phases obtains better performance for shorter average Manhattan distance.

6.5.3 Scalability

Figure 6.17 shows the time against the number of processors for routing MCC2-45 using prog5 in a logarithmic scale. There is a high degree of parallelism inside the program since increasing number of PEs from 16 to 256 for MCC2-45 speeds up the program

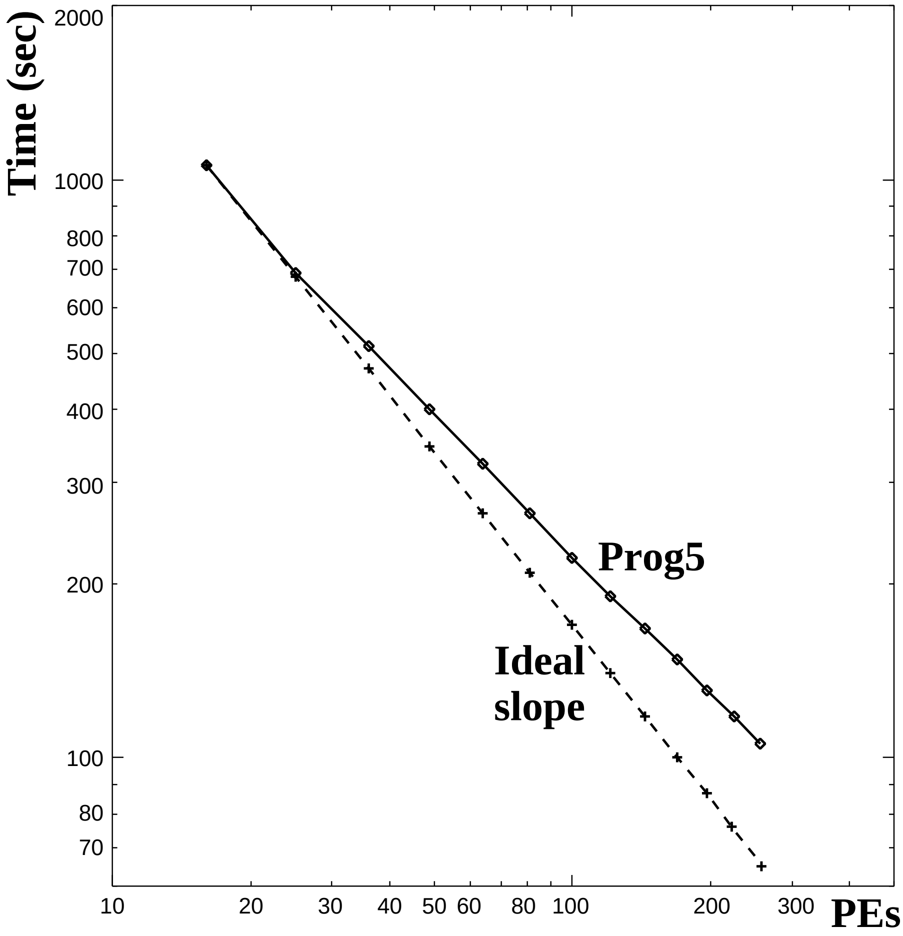


Figure 6.17: Time against number of PEs for MCC2-45.

by 10 times. We have also a good scalability since the speed up is not saturated yet even though we use up to 256 PEs

Chapter 7

Conclusion

A new parallel wire routing algorithm has been introduced. This router consists of two phases with two different grid assignments to decrease the communication cost. The whole routing area is divided among the PEs to decrease the required memory for each PE. The first phase rotates the assignment area of every PE to route all short nets using only few messages. The second phase divides the grid into slices to increase the parallelism among nets and to decrease the number of messages.

A cooperative relation between global and detailed routings is introduced to improve the successful routing ratio. The nets are divided into groups. After finding the detailed paths for one group, the actual partitions capacities are calculated by the detailed router to be used for the global routing of the next group.

A new algorithm for detailed routing called max-cap algorithm is also introduced to improve the successful routing ratio. In this algorithm, the maximum capacities for the slices are obtained, and each net is routed by a path which decreases this capacity only by one. Max-cap algorithm obtains higher routing ratio than the well known maze running algorithm.

Several programs are implemented on the AP1000 parallel computer and evaluated using MCM benchmarks. The results show that our router has a high degree of parallelism and low communication cost. Using the max-cap algorithm and the actual capacity (cooperative global and detailed) improves the successful routing ratio at the expense of execution time as dividing nets into groups decreases the parallelism among nets. Using two phases decreases the communication cost and improves parallelism without affecting the successful routing ratio.

In the future work, we will check the following points:

- Effect of using dynamic load balance on the speed of the Amon. This will increase the communication cost but it will improve the load balance.

- In Max-cap algorithm, to route each net, we find the maximum number of paths which can be used to route this net and then choose one path for this net. This is meant that we have to recalculate maximum number of paths for each net. The speed of the program could be improved if we collect more than one net which will be routed within the same slice in a group, find the maximum number of paths in this slice, then choose paths for all nets in this group. That is meant to calculate maximum number of paths only once per group.

- Moving (changing) the boundaries between slices may also improve the successful routing ratio. If a boundary between two slices is removed (two slices become one slice), a new path may be found. We will try to move the boundaries between slice by the same idea of the rotating area algorithm.

Bibliography

[1] P. Banerjee, "Parallel Algorithms for VLSI Computer Aided Design", PTR Prentice Hall 1994.

[2] R. Venkateswaran and P. Mazumder, "Coprocessor Design for Multilayer Surface Mounted PCB Routing", IEEE Trans. on VLSI, Vol. 1, No. 1 March 1993, pp 31-45, 1993.

[3] M.J.Lorenzetti and D.S.Baeder, "Physical Design Automation of VLSI Systems", Edited by B.T.Preas and M.J.Lorenzetti.

[4] C. Y. Lee, "An algorithm for path connections and its applications", IRE Trans. on Electronic Computers, vol. EC-10, pp 346-365, 1961.

[5] K.Suzuki, Y.Matsunaga, M. Tachibana, and T. Ohtsuki, "A Hardware Maze Router with Application to Interactive Rip-Up and Reroute", IEEE Trans. on CAD, vol. CAD-5, pp 466-476, 1986.

[6] K. Suzuki, T. Ohtsuki, and M.Sato, "A Gridless Router : Software and Hardware Implementation", VLSI 87, pp 153-163, 1987.

[7] T. Yamauchi, A. Ishizuka, T. Nakata, N. Nishiguchi, and N. Koike, "PROTON : A parallel router on an MIMD parallel machine", ICCAD, pp 340-343, 1991.

[8] H. Date, Y. Matsumoto, K. Kimura, K. Taki, H. Kato, and M. Hoshi, "LSI-CAD Programs on Parallel Interface Machine", Proc. International Conference on fifth generation computer system 1992, pp 237-247, 1992.

[9] H. Date and K. Taki, "A Parallel Lookahead Line Search Router with Automatic Ripup and Reroute", Proc The European Conference on Design Automation, pp 117-121, 1993.

[10] Y. Takahashi, "Parallel automated wire routing with a number of competing processors", International Conference on Supercomputing, ACM vol.18, Number 3, pp 310-317, 1990.

[11] Y. Takahashi, "Parallel maze running and line search algorithms for LSI CAD on binary tree multiprocessors", Word conference on information/comm. Seoul 89, pp 128-136, 1989.

[12] Y .Won and S. Sahni, "Maze routing on a hypercube multiprocessor computer", Proc. Int. Conf. Parallel Processing, pp 630-637, 1987.

[13] O.A.Olukotun and T.N.Mudge, "A preliminary investigation into parallel routing on a hypercube computer", Proc. Design Automation Conf.78, pp 814-820, 1987.

[14] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A new technique to improve parallel automated single layer wire routing", Proc. Performance evaluation of parallel systems 1993, PEPS'93, pp 134-141, 1993.

[15] Shimamoto, Hane, Shirakawa, Tsukiyama, Shinoda, Yui and Nishiguchi, "A distributed routing system for multilayer SOG", Proc. of EURO-DAC 92, pp 298-303, 1992.

[16] D.I.Moldovan, "Parallel Processing From Applications to Systems", Morgan Kaufmann Publishers, 1993.

[17] G.Langholz, J.Francioni, and A.Kandel, "Elements of Computer Organization", Publisher: Prentice-Hall Inc., 1989.

[18] J.M.Ortega, "Introduction to Parallel and Vector Solution of Linear Systems", Publisher: Plenum Press, Newyork, 1988.

[19] Kai Hwang, "Advanced Computer architecture with parallel programing", Publisher: McGraw-Hill, Inc., Preliminary edition 1993.

[20] S.A.Wiliams, "Programming Models for Parallel systems", Publisher: John Wiley & Sons Ltd, 1990.

[21] H.Ishihata, T.Horie, S.Inano, T.Shimizu, S.Kato, and M.Ikesaka , "Third generation message passing computer AP1000", International symposium on supercomputing, pp 46-55, 1991.

[22] T.Ohtsuki, "Maze running and line search algorithms", Layout design and verification 1986, North-Holland , pp 99-131 , 1986.

[23] M. Hanan, "On Steiner's problem with rectilinear distance", SIAM Journal of Applied Mathematics, 14, pp 255-265, March 1966.

[24] F. Hwang, "An O(nlogn) algorithm for suboptimal rectilinear Steiner trees", IEEE Trans. on Circuits and Systems, CAS-26, pp 75-77, 1978.

[25] S.B.Akers, "Routing", Design Automation of Digital systems: Theory and Techniques, vol. 1, pages 283-333, Prentice Hall, Editor: M.A.Breuer, 1972.

[26] T.Watanabe, H.Kitazawa, and Y.Sugiyama, "A parallel adaptable routing algorithm and its implementation on a two dimensional array processor", IEEE Trans. on Computer aided design, Vol. CAD-6, No. 2, pp 241-250, 1987.

[27] K.Mikami and K.Tabuchi, "A computer program for optimal routing of printed circuit connections", IFIPS Proc., vol. H47, pp 1475-1478, 1968.

[28] D.W.Hightower, "A solution to line routing problem on a continous plane", Proc. 6th Design Automation Workshop, pp 1-24, 1969.

[29] J. H. Hoel, "Some Variations of Lee's algorithm", IEEE Trans. on computer vol. C-25, pp 19-24, 1976.

[30] J. Soukup, "Fast maze router", Proc. Design Automation Conf. 78, pp 100-102, 1978.

[31] J. Soukup, "Maze router without a grid map", ICCAD 92, pp 382-385, 1992.

[32] S.B.Akers, "A modification of Lee's path connection algorithms", IEEE trans. on Electronic Computers (short notes), vol. EC-16, pp 97-98, 1967.

[33] E. S. Kuh and M. Marek-Sadowska, "Global Routing", Layout design and verification 1986, North-Holland , pp 169-198, 1986.

[34] M. Burstein and R. Pelavin, "Hierarchical wire routing", IEEE Trans. on Computer Aided Design, CAD-2, pp 223-337, 1983.

[35] M. Marek-Sadowska, "Global router for gate array", IEEE proc. Int. Conf. on Computer Design 84, pp 332-337, 1984.

[36] R.J.Brouwer and P.Barerjee, "PHIGURE: A Parallel Hierarchical Global router", Proc.27th Design Automation Conf., pp 360-364, June 1990.

[37] J.Rose, "Locusroute: A Parallel global router for standard cells", Proc. Design Automation Conf. 88, pp 189-195, 1988.

[38] J.Rose, "Parallel global routing for standard cells", IEEE Trans. Computer-aided Design, pp 1085-1095, Oct. 1990.

[39] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A Parallel Slice Maze Router", International Symposium on Fifth Generation Computer Systems 1994, FGCS94 LSI-CAD, W6 pp 67-73, 1994

[40] M. Goda, N. Toyama and T. Watanabe, "A parallel detailed router TRED", Proc. of Transputer/Occam Japan 1994.

[41] Kei-Yong Khoo and J. Cong, "A fast multilayer general area router for MCM designs", IEEE trans. on Circuits and Systems, Vol. 39, No. 11 Nov 1992, pp 841-851. 1992.

[42] Kei-Yong Khoo and J. Cong, "An efficient multilayer MCM router based on four via routing", DA 30, pp 590-595, 1993.

[43] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "Amon : A parallel slice algorithm for wire routing", Proc. International Conference on Supercomputing 1995, ICS'95, pp 200-208, 1995.

List of Publications by the Author

Major publications

[1] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A new technique to improve parallel automated single layer wire routing", Proc. Performance evaluation of parallel systems 1993, PEPS'93, pp 134-141, 1993.

[2] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A Parallel Slice Maze Router", Proc. International Symposium on Fifth Generation Computer Systems 1994, FGCS94 LSI-CAD, W6 pp 67-73, 1994

[3] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "Amon : A parallel slice algorithm for wire routing", Proc. International Conference on Supercomputing 1995, ICS'95, pp 200-208, 1995.

[4] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A two phases, cooperative detailed/global parallel wire routing algorithm", Submitted to Trans. of Information Processing Society of Japan.

[5] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "Amon2: A parallel wire routing algorithm on a torus network parallel computer", Accepted in International Conference on Supercomputing 1996.

Oral Presentations

[1] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A new technique to improve parallel automated single layer wire routing", Proceeding of the Second Parallel Computing Workshop 1993, Kawazaki, Japan, PCW93, pp P1R1- P1R8.

[2] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A Parallel Automated Wire Router", Technical report of IEICE, SWOPP 94, CPSY94-43, pp 17-24, 1994.

[3] H. Keshk, S. Mori, H. Nakashima and S. Tomita, "A Parallel Router on AP1000", Proceeding of the Third Parallel Computing Workshop 1994, Kawazaki, Japan, PCW94, pp P2Q1- P2Q8.