Sequential/Parallel Global Routing Algorithms for VLSI Standard Cells

A Thesis

Presented to

The Faculty of Graduate Studies

of

The University of Guelph

by

HAO SUN

In partial fulfilment of requirements

for the degree of

Masters of Science

April, 2004

©Hao Sun, 2004

ABSTRACT

Sequential/Parallel Global Routing Algorithms for VLSI Standard Cells

Hao Sun
University of Guelph, 2004

Advisor:
Professor Shawki Areibi

Global routing is an important and time consuming step in VLSI physical design automation. In order to effectively solve this problem, three standard cell global routers are presented in the form of (i) a Sequential Heuristic Global Router (SHGR) (ii) a Hierarchical Heuristic Global Router (HHGR) and (iii) an Integer Linear Programming (ILP) based Global Router (ILP-GR).

The objective of SHGR is to find the minimum cost path for each net by enumerating a set of possible 2-bend routes. It achieves 16% improvement over wirelength obtained by placement based on Half Perimeter Wire Length (HPWL) method. HHGR is a pre-processing technique that provides good routing estimations in reasonable time. It can predict the routability of circuits and reduces CPU time on average by 82.5% compared with SHGR. ILP-GR formulates the global routing problem as two Integer Linear Programming (ILP) models. These ILP models are further relaxed to Linear Programming (LP) models to reduce computation time. ILP-GR overcomes the net-ordering problem and provides optimal solutions which can be used as upper bounds for other global routers.

It achieves on average an improvement of 6.7% over wirelength solutions produced by the SHGR technique.

In addition, SHGR and ILP-GR are also parallelized by using the Message Passing Interface (MPI) library to speed up the running time and achieve better performance. The parallel algorithms are evaluated on a distributed network of Sun-blade 2000 workstations. Experimental work shows promising results where good speedups are obtained and the total channel density reduced on average by 1.59% through parallel implementations.

Acknowledgements

My sincere thanks go to Dr. Shawki Areibi for his support and advice throughout this research. Without his help, this work would never have been possible.

Thanks should also go to Sharcnet for the financial and technical support.

To

my family

whose love and encouragement helped accomplish this thesis.

Contents

1 Introduction **1**

1.1 Overview of VLSI Design Process . . . 2

1.2 Research Motivations . . . 4

1.3 Overview of the Approach . . . 8

1.4 Contributions . . . 9

1.5 Thesis Organization . . . 10

2 Background **11**

2.1 Physical Design and Circuit Layout . . . 11

2.1.1 Layout Strategies and Styles . . . 15

2.2 Graph Theory . . . 18

2.2.1 Grid Graph model . . . 20

2.2.2 Connection Pattern . . . 22

2.2.3 Minimum Spanning Tree (MST) . . . 25

2.2.4 Netlist and Admissible Routes . . . 27

2.3 Global Routing for Standard Cell Design . . . 28

2.4 Parallel Programming . . . 31

2.4.1 Classification 31

2.4.2 Shared Memory and Distributed Memory 33

2.4.3 Amdahl's Law and Parallel Speedup 33

2.5 MCNC Benchmarks 34

2.6 Summary 36

3 Literature Review 37

3.1 Sequential Approaches 38

3.1.1 Two-terminal Sequential Algorithms 39

3.1.2 Multi-terminal Algorithms 42

3.2 Concurrent Approaches 47

3.2.1 Hierarchical Based Approaches 47

3.2.2 Integer Programming Based Approaches 51

3.3 Summary 54

4 Heuristic/ILP-based Routing Algorithms 55

4.1 A Sequential Heuristic Global Router (SHGR) 56

4.1.1 Cost Array Routing Model 56

4.1.2 SHGR Implementation 57

4.1.3 Experimental Results 61

4.2 A Hierarchical Heuristic Global Router (HHGR) 64

4.2.1 HHGR Implementation 65

4.2.2 Experimental Results 67

4.3 An ILP Based Global Router (ILP-GR) 69

4.3.1 Minimal Rectilinear Steiner Tree (MRST) 70

4.3.2 ILP Formulation 74

4.3.3 Linear Relaxation 83

4.3.4 Parameter Tuning 86

4.4 Summary 93

5 Parallel Implementation 95

5.1 Parallel Implementation of SHGR 96

5.1.1 Distribution of Cost Array 98

5.1.2 Wire Assignment Strategy 99

5.1.3 Cost Array Update 100

5.2 Parallelization of ILP-GR 101

5.2.1 Parallel Implementation 102

5.2.2 Wire Assignment Strategy 102

5.3 Experimental Results 106

5.4 Summary 108

6 Conclusions and Future Directions 110

6.1 Conclusions 111

6.2 Future Work 113

A Glossary 114

B CPLEX Solver 116

B.1 Introduction to CPLEX 116

B.2 Using CPLEX to Solve ILP/LP Problems 117

C MPI **120**

C.1 Introduction to MPI 120

C.2 MPI Datatypes and Basic Functions 120

C.3 MPI Core Functions 121

Bibliography **124**

List of Tables

2.1 MCNC benchmarks 35

2.2 The chip dimension of MCNC benchmark circuits 35

4.1 Wirelength and time of SHGR 62

4.2 Wire length comparison between the placement CAD tool and the SHGR 62

4.3 CPU Time comparison between SHGR and HHGR 67

4.4 Wirelength comparison between SHGR and HHGR 67

4.5 Number of pins in the nets for MCNC benchmark circuits 73

4.6 Comparison of $Steiner\_ILP1/Steiner\_ILP2$ to generate Steiner Trees 74

4.7 Comparison of wire length between Model 1 and Model 2 82

4.8 Comparison of the number of tracks between Model 1 and Model 2 83

4.9 Comparison of net conneciton ILP/LP models 86

4.10 Comparison of channel density ILP/LP models 87

4.11 Comparison of Dual Simplex and Primal-Dual Barrier Optimizers . 91

4.12 Wirelength comparison between SHGR and ILP-GR 91

4.13 CPU time comparison of SHGR and ILP-GR 93

5.1 Speedup of the parallel heuristic router 106

5.2 Speedup of the ILP based router 106

5.3 Parallelism improves the total density 107

C.1 The basic MPI datatypes 121

List of Figures

1.1 VLSI Design Flow 2

1.2 Interconnection delay. 5

1.3 Power consumption related to interconnection capacities. 6

1.4 Solution strategy for Global Routing. 8

2.1 Physical design cycle 12

2.2 Multilevel cluster hierarchy 13

2.3 Terminology for VLSI routing problem. 14

2.4 Difference between global routing and detailed routing 15

2.5 Gate-Array layout and Standard-Cell layout 16

2.6 General-Cell layout and Full-Custom layout 18

2.7 Illustration of a graph G=(V,E) 19

2.8 Layout of a chip with a grid-graph to represent the modules. 20

2.9 A standard cell design circuit and its coarse grid graph representation. 22

2.10 Interconnection patterns 23

2.11 Approximation wire estimation and congestion 24

2.12 A Minimum Spanning Tree Algorithm 25

2.13 An example of MST construction 26

2.14 A netlist consists of two nets. 27

2.15 Channel density and total density. 29

2.16 Feedthrough cell and wire 30

2.17 Example of two nets that span multiple rows. 31

2.18 Parallel Models 32

3.1 Methodologies for global routing. 38

3.2 Maze routing algorithm 39

3.3 Line probe routing algorithm 41

3.4 The Hanan grid and the set of Hanan points. 43

3.5 Hanan's theorem. 44

3.6 Comparison of the rectilinear in 2-geometry and in 4-geometry . . 45

3.7 A floorplan and its preprocessed cut-tree 48

3.8 Pin configuration and net types in 2x2 cell array. 50

4.1 Cost Array model 57

4.2 Sequential Heuristic Algorithm 58

4.3 A 5-point net is decomposed into four 2-point segments 59

4.4 Four Possible Permutations Between Cluster A and Cluster B . . . 59

4.5 2-bend Algorithm to find routes between point (0,0) and (C,R) . . 60

4.6 2-bend Routes in a 2x3 Grid Graph 61

4.7 Wirelength comparison of SHGR and HPWL 63

4.8 A 3 level cluster seed placement 64

4.9 Hierarchical heuristic algorithm 65

4.10 A net is cut into segments 66

4.11 CPU time comparison of SHGR and HHGR 68
4.12 Wirelength comparison of SHGR and HHGR 69
4.13 Minimum Spanning Tree (MST) and Minimum Rectilinear Steiner Tree (MRST). 70
4.14 Algorithm to find a set of Steiner trees 71
4.15 Function Find_SteinerTree() 72
4.16 Example of the net connection model 77
4.17 Example of the channel density model 81
4.18 Wirelength comparison of SHGR and ILP-GR 92
4.19 CPU time comparison of SHGR and ILP-GR 94

5.1 Parallel heuristic algorithm 97
5.2 Distribution of the Cost Array 99
5.3 Wire assignment strategy 100
5.4 The parallel implementation in the whole global routing process . . 103
5.5 ILP based parallel algorithm 104
5.6 Center partition wire assignment 105
5.7 Speedup obtained by parallelizing SHGR on large benchmark circuits 107
5.8 Speedup obtained by parallelizing ILP-GR on large benchmark circuits108

Chapter 1

Introduction

The last few decades brought explosive growth in the electronics industry due to the rapid advances in integration technologies and the different benefits of large-scale system design. Integrated circuits today consist of hundreds of millions of transistors. This manufacturing capability and complexity, combined with the economic benefits of large electronic systems, are forcing a revolution in the design of these systems and challenging system designers who are involved in the design of integrated circuits. Due to the tremendous increase in complexity, automating the design process has become a crucial issue [HU85a].

The phrase associated with the task of automatically designing a circuit using Computer Aided Design (CAD) tools is called Design Automation (DA). The objective of the DA research field is to fully automate the tasks of designing, verifying, and testing a circuit. For a complicated problem in the modern VLSI design, an appropriate approach is to use a divide-and-conquer strategy in which the whole design task is broken down into several sub-tasks [Sher99]. These sub-tasks are then more manageable to be solved using mathematical and heuristic techniques.

1.1 Overview of VLSI Design Process

The VLSI design process is generally divided into a sequence of phases: *System Specification, Functional Design, Logic Design, Circuit Design, Physical Design, Fabrication and Testing.* These phases are illustrated in Figure 1.1 and are briefly described below [Sher99].

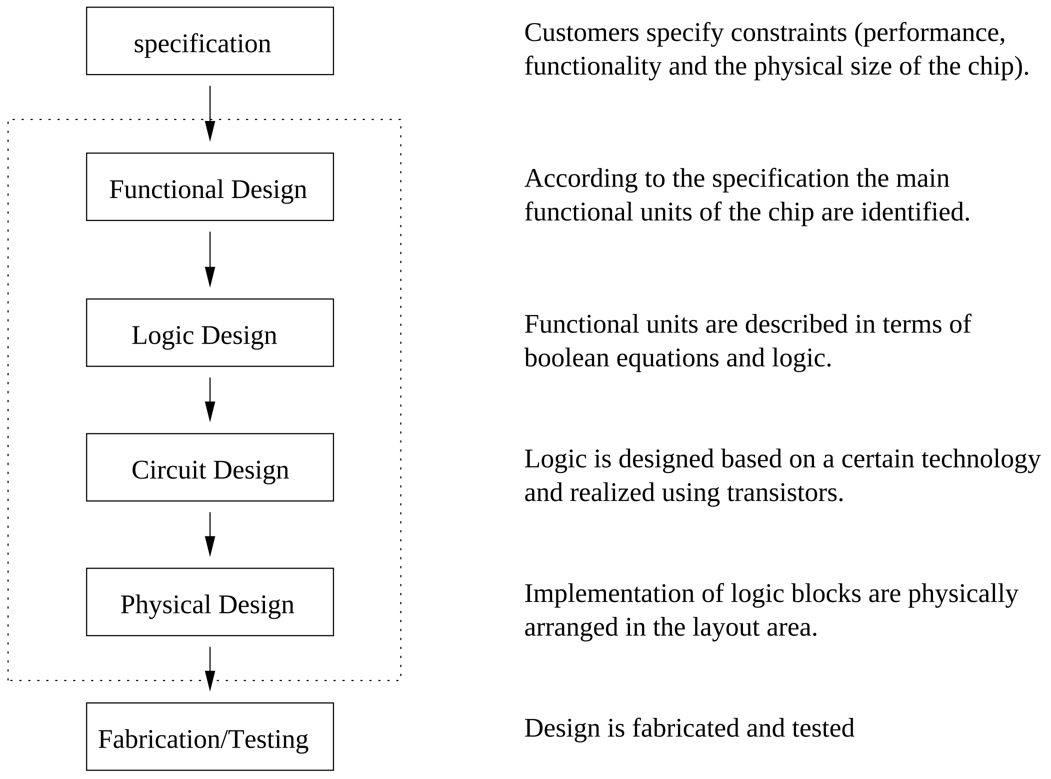

specification
Customers specify constraints (performance, functionality and the physical size of the chip).
Functional Design
According to the specification the main functional units of the chip are identified.
Logic Design
Functional units are described in terms of boolean equations and logic.
Circuit Design
Logic is designed based on a certain technology and realized using transistors.
Physical Design
Implementation of logic blocks are physically arranged in the layout area.
Fabrication/Testing
Design is fabricated and tested

Figure 1.1: VLSI Design Flow

- **Specification**: Starts with a set of requirements for the circuit, which includes what the system will do, how the system will be divided into components and how these components work together. This process outputs specifications for the size, speed, power and functionality of the circuit.

- **Functional Design**: Identifies and estimates the interconnect requirements between the units, area, power and other parameters of each component.

- **Logical Design**: Decides how each component of the system will be expressed logically. Hardware description languages (HDL), such as VHDL and Verilog, are generally used to give the structural/behavioral description.

- **Circuit Design**: Maps the logic blocks to available physical circuit blocks in the circuit topology and creates a technology-dependent description of the circuit. At this level, the whole circuit is implemented as transistors.

- **Physical Design**: Physically realizes the circuit by converting the circuit representation of each component into a geometric representation (also called a layout). Connections between different components are also expressed as geometric patterns. The end result of physical design is a placed and routed design, from which the photolithography masks can be derived for chip fabrication. The physical design problem is an NP-hard problem, therefore it is usually broken down into several sub-problems, referred to as partitioning, floorplanning, placement and routing. This thesis is mainly concerned with the global routing problem that follows module placement.

- **Fabrication and Testing** : In the final step of VLSI design the circuit is tested to ensure that all criteria are satisfied, the wafer is manufactured and the chip is packaged and tested.

1.2 Research Motivations

Over the past few decades, fabrication technique advances have decreased transistor feature size in VLSI circuits significantly, meanwhile, the scale of integration has increased dramatically. These changes lead to a corresponding increase in the number of interconnections inside a chip. These interconnections occupy the valuable chip area resources and consequently degrade the chip performance. Global routing is an extremely important and time consuming phase in the VLSI physical design cycle. For a VLSI circuit containing over 100,000 cells and nets, global routers can easily take many hours to route nets within the circuit.

The first motivation of this thesis is to develop efficient global routing algorithms, which can produce near optimal solutions that tend to minimize the routing area and improve the circuit performance. In VLSI standard cell design, the total area of a chip consists of the active area used by cells, the routing area, the unused area and the pin area. Normally unused area and pin area have minimal effects on chip size. Therefore, reduction in chip size is only possible by reducing the active area and routing area. The minimization of routing area is crucial due to the following factors:

- **Yield and Cost:** The minimization leads to smaller die sizes, which can result in a larger number of dies per wafer and increase the percentage utilization of a wafer. The cost of a chip is dependent on the yield, which depends on the die sizes.

- **Delay:** Delay is one of the most important factors to affect the performance of a chip. With the technology moving to deep sub-micron design rules, the interconnection delay has become the dominant factor in determining the circuit speed [Kang03]. It has been reported that the interconnection delay can consume from 50% to 70% of the clock cycle in many cases [Bako90]. Therefore, a reduction in the interconnection delay will have an important impact on the overall performance of the circuit, as illustrated in Figure 1.2. Most of the previous researches aimed at yielding the minimum chip area and the shortest wire length are usually referred to as area-driven global routing. However, in submicron chip design, shorter wire length does not yield better performance. This is due to fact that the delay at the sink pins depends on the distance between the source and sink pins as well as the total wire length.

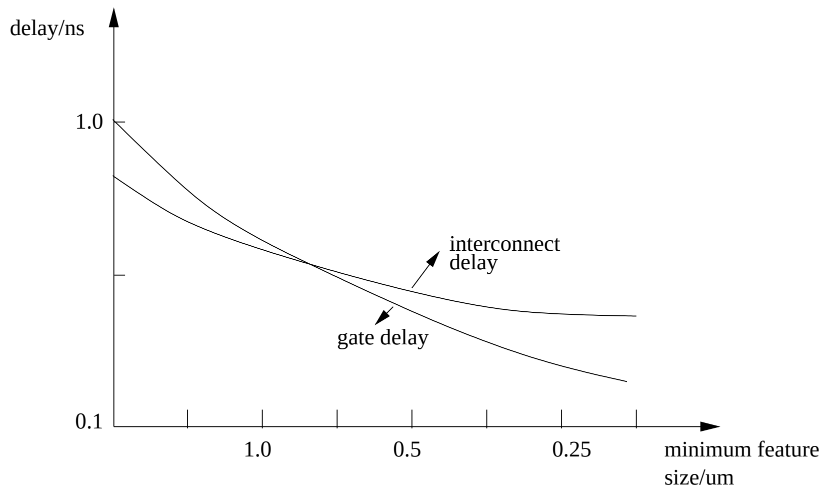


Figure 1.2: Interconnection delay.

- **Power Consumption:** The increasing prominence of portable systems and wireless applications require low power consumption of chips [SKan03]. The average power consumption in conventional COMS digital circuits can be expressed as follows:

$$P = \alpha_T \cdot C_{load} \cdot V_{DD}^2 \cdot f_{CLK} \tag{1.1}$$

where C_{load} includes internal capacitances associated with the gate's transistors and external capacitances related to wire interconnections. With the technology advancing to sub-micron design, the leakage power that is related to the interconnection capacities has become the dominant factor in determining the overall power consumption, as seen in Figure 1.3 [Horg04]. Therefore, the need to limit power consumption can be achieved by reducing wire interconnections.

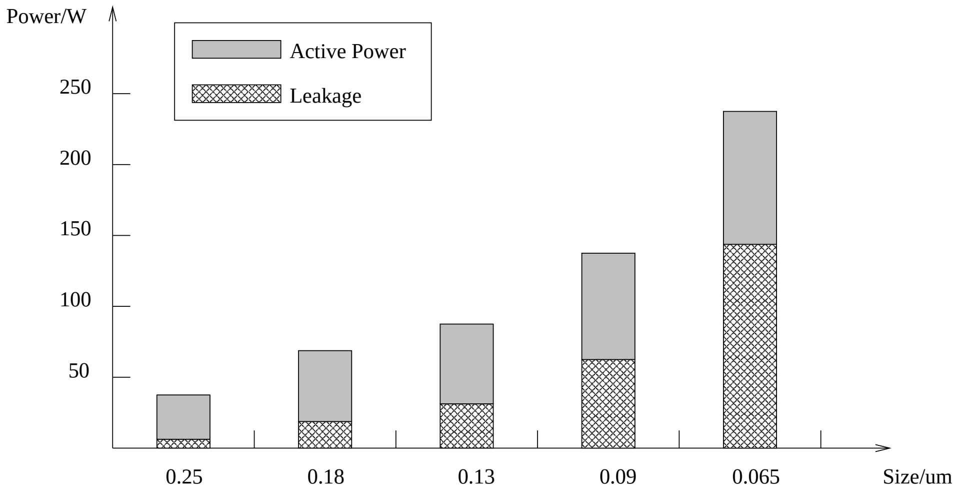


Figure 1.3: Power consumption related to interconnection capacities.

- **Congestion:** Congestion is widely addressed in global routing algorithms [Kris02], since a congested area can degrade the performance of the global router and even yield unroutable results. In global routing, congestion and delay are two competing objectives [Hu00].

The second motivation is to make use of the current high-performance hardware to solve complex and large global routing problems efficiently. Sequential based CAD algorithms have been successfully used in VLSI design for many year. However, with the increasing complexity of the VLSI chips, conventional sequential approaches face difficulty to follow this trend. Since using high-performance multiprocessor systems is becoming more feasible for researchers, parallel algorithms are put forward to meet the new challenges in VLSI design field. The necessity of parallel algorithms can be attributed to the following:

- **Shortening the Runtime:** It is normal for a circuit today to have over 100 million transistors and gates. The large number of transistors and gates become a burden on sequential CAD tools. Since parallel processing can reduce computation time dramatically, it can be used extensively for VLSI CAD based applications.

- **Solving Larger Sized Problem:** Since the number of gates on a chip is increasing, conventional sequential CAD tools will reach at some point a memory limitation. However, parallel algorithms handle this kind of problem by distributing the computation among several processors.

- **Achieving Better Quality:** VLSI CAD problems are mainly formulated as combinatorial optimization problems that are NP-complete [Kahn95]. Therefore, heuristics are often used to solve these problems. By using parallel algorithms, it has been observed that better quality results can be obtained [Bane94].

1.3 Overview of the Approach

In order to effectively solve the global routing problem, three standard cell global routing techniques are presented in the form of (i) a heuristic approach, (ii) an Integer Linear Programming (ILP) based approach and (iii) a hierarchical based approach. As illustrated in Figure 1.4, the heuristic approach and the ILP based approach utilize solutions obtained from the flat placement approach. They are further parallelized to speed up the running time and achieve better performance. The hierarchical approach on the other hand transforms solutions obtained from hierarchical placement solution into routing estimations of reasonable quality.

Experimental results show excellent performance from all three global routers. In addition, good speedups and solution qualities are obtained from the parallel implementation. The average speedup achieved is 5 on six processors, and the total density is reduced by an average of 1.59% as well.

1.4 Contributions

The main contributions of this dissertation are summarized as follows:

- Explore the feasibility of a sequential heuristic approach to effectively solve global routing for large circuits in reasonable time [Sun04].

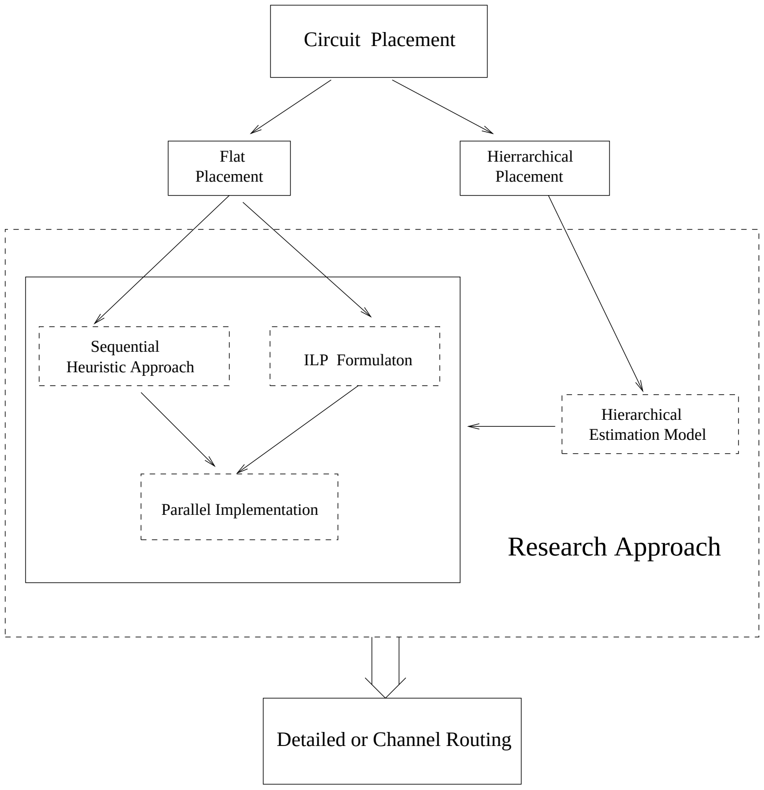


Figure 1.4: Solution strategy for Global Routing.

- Investigate an effective generation mechanism of Minimal Rectilinear Steiner Trees (MRST) that is further used by the proposed global routing techniques.

- Development of ILP-based models which produce optimal solutions and accordingly overcome the net-ordering problem. Solutions obtained by these models can be utilized as upper bounds on results obtained by heuristics.

- Intelligent parallel implementation of the above mentioned algorithms to solve large benchmarks and reduce the execution time.

- Development of a novel hierarchical estimation global routing technique based on hierarchical placement to predict routability for VLSI ASIC design.

1.5 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 introduces the necessary background that defines the concept of global routing for standard cell design. It also briefly reviews graph theory and parallel programming techniques. Chapter 3 reviews previous work on the global routing problem, including sequential and concurrent approaches. In chapter 4, a Sequential Heuristic Global Router (SHGR), a Hierarchical Heuristic Global Router (HHGR) and an Integer Linear programming (ILP) based Global Router (ILP-GR) are implemented. In chapter 5, the parallel implementations of SHGR and ILP-GR are presented. Finally, chapter 6 provides conclusions and future work.

Chapter 2

Background

This chapter gives a detailed background on physical design automation in general and global routing in particular. It also reviews graph theory and parallel programming techniques. In addition, several test circuit benchmarks [Kozm91] are introduced at the end of this chapter.

2.1 Physical Design and Circuit Layout

Physical design automation is the process of mapping structural representations into layout representation of circuits. Structural representations describe the system in terms of logic components and their interconnects. Layout representations define circuits in terms of a set of geometric objects which specify the dimensions and locations of transistors and wires on a silicon wafer. As seen in Figure 2.1, physical design is usually divided into the following steps:

- **Partitioning**: In this phase, a circuit is partitioned into several smaller

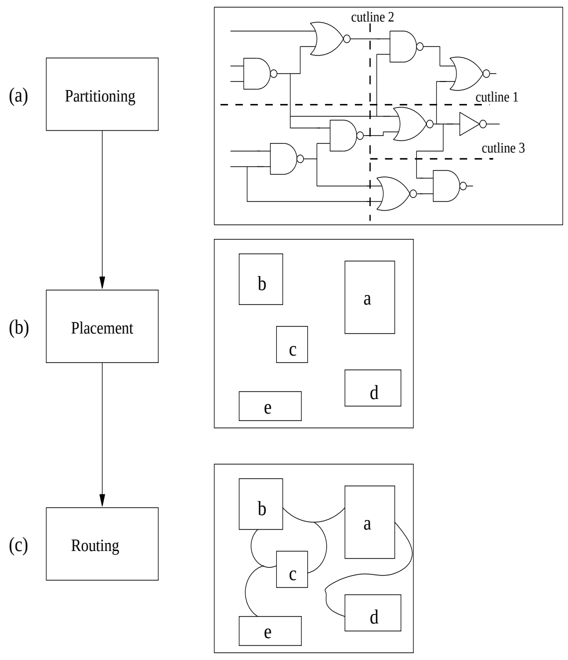


Figure 2.1: Physical design cycle

blocks/sub-circuits according to their features and connectivity. The objective of partitioning is to reduce the complexity of mapping a circuit onto Multiple Chip Modules (MCM). It is also used as a preprocessing stage for circuit placement [Sher99]. Figure 2.1(a) shows that the input circuit is partitioned into five blocks/sub-circuits with several nets cut between partitions.

- **Floorplanning and Placement**: The modules in a chip may have variable shapes and dimensions. The positioning or placement of these variable modules onto the silicon wafer is called floorplanning [Leng90a]. Placement is the process of assigning modules to the silicon wafer such that the wire lengths between modules and the area of the chip is minimized. Figure 2.1(b) shows that the five blocks (a, b, c, d and e) obtained from partitioning phase are placed onto the circuit.

Both flat and hierarchical placement approaches are used for circuit placement [Yang03]. Hierarchical placement approaches can significantly reduce the complexity of the circuit and improve the performance of the design process. Circuit clustering plays a fundamental role in hierarchical designs. Clustering in steps tends to reduce the circuit size gradually by adding intermediate levels to the hierarchy. This technique is often called "multi-level" or "hierarchical" clustering.

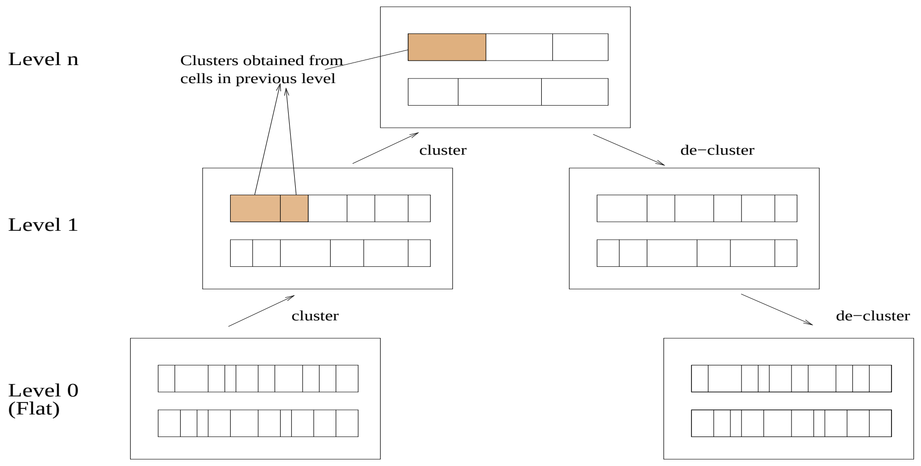


Figure 2.2: Multilevel cluster hierarchy

As seen in Figure 2.2, multi-level clustering is a two-step procedure, first proceeding bottom-up, and then top-down. The bottom-up procedure is a clustering phase that involves the grouping of highly connected cells. The top-down procedure then determines the location for all the clusters, and consequently the location of cells within those clusters [Kary97].

- **Routing**: Following the placement phase, the pins belonging to different nets have to be connected according to certain criteria. In general, the routing phase is to map these netlists into an actual physical geometry. The main objective of this phase is to minimize the total routing area without violating the constraints [Sher99]. Figure 2.3 illustrates some terms used for the routing problem. The top and the bottom rows are called "*top boundary*" and "*bottom boundary*". The term "*channel*" refers to the area between the two boundaries. In a net, the horizontal segment is called a "*trunk*" and the vertical is called "*branch*". The horizontal line on the circuit along which a trunk is placed is called a "*track*". A "*via*" is used to connect a wire segment in two different layers.

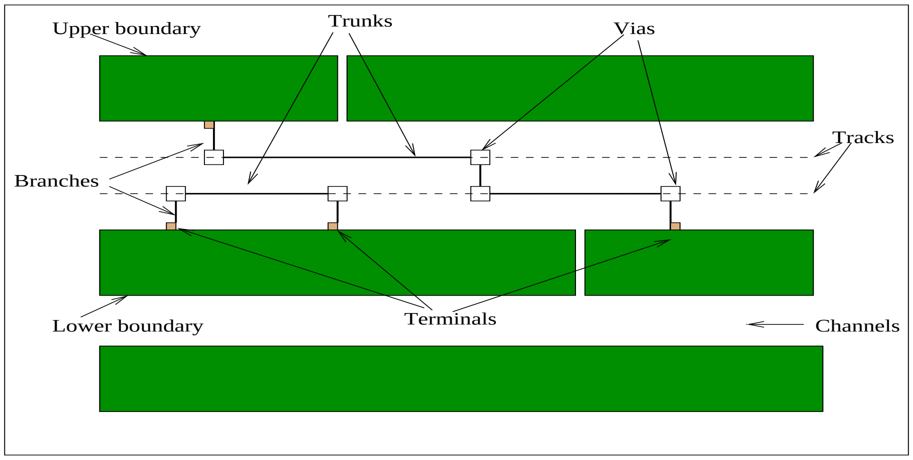


Figure 2.3: Terminology for VLSI routing problem.

Routing is normally performed in two stages. The first stage, called **global routing**, determines the channels through which a connection will run. As illustrated in Figure 2.4 (a), global routing assigns $channel_1$ to Net_1 and $channel_2$ to Net_2. The second stage, called **detailed routing**, fixes the precise paths that a wire will take. As illustrated in Figure 2.4 (b), the exact geometric layout (tracks and branches) is given within the assigned channels. In general, global routing decomposes a large routing problem into small, manageable problems for detailed routing.

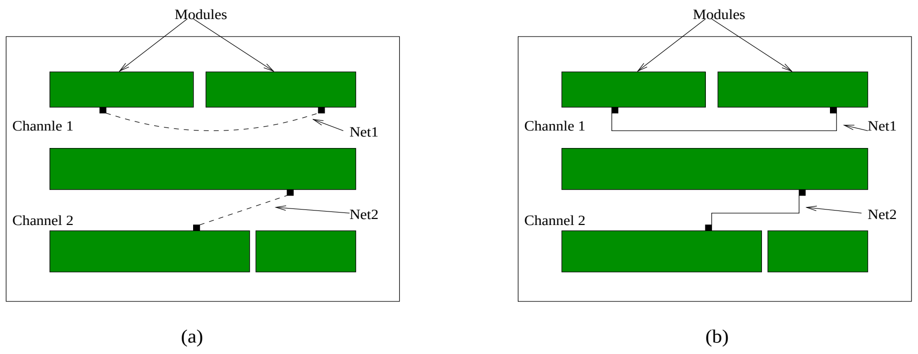


Figure 2.4: (a) Global routing. (b) Detailed routing.

Global routing is an extremely important and time consuming phase in the VLSI physical design cycle. The routing time complexity depends heavily on layout strategies which will be introduced next.

2.1.1 Layout Strategies and Styles

Physical design is an extremely complex process and even after breaking the entire process into several conceptually easier steps, it has been shown that each subtask is computationally NP-hard [Hana85]. However, market requirements demand a quick time-to-market and high yield. As a result, restricted models and design styles are used in order to reduce the complexity of VLSI physical design. The

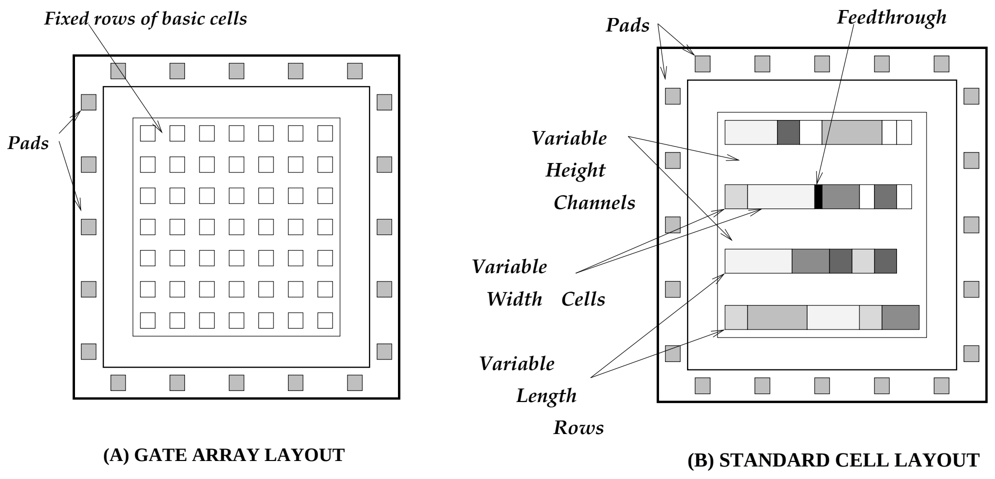


Figure 2.5: Gate-Array layout and Standard-Cell layout

classification and comparison of layout styles is given in [Ueda86]. Currently, the most popular VLSI physical design styles are *gate-array*, *standard-cell*, *general-cell*, and *full-custom design*.

2.1.1.1 Gate-Array Design

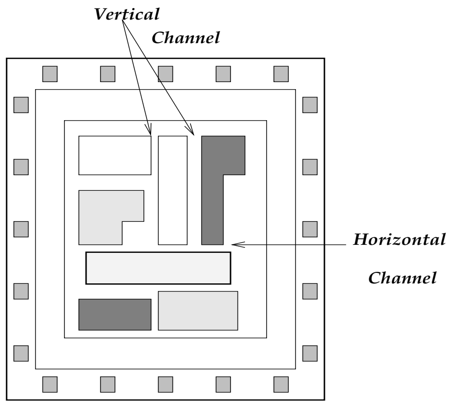
In gate-array design, the entire wafer is prefabricated with an array of identical gates or cells. As shown in Figure 2.5 (a), the cells are separated by both vertical and horizontal spaces called vertical and horizontal channels. The name "gate-array" signifies the fact that each cell may simply be a gate, such as a 2 input OR gate [SKan03]. A special case of the gate array is the *Field Programmable Gate Array* (FPGA) topology. In FPGA technology, all wires and interconnections are manufactured on the chip, and programmable switches are fabricated into the interconnections. The desired design can be implemented by programming the interconnections between the wires and gates. Since the entire physical chip is pre-fabricated, the turn-around time is fast. It is also well suited for automated design due to its highly regular layout style. However, FPGAs are not space-efficient since all the wires and switches manufactured are generic to allow different interconnections. Furthermore, current routing technology used by FPGAs adds a significant delay to interconnections [Babb93].

2.1.1.2 Standard-Cell Design

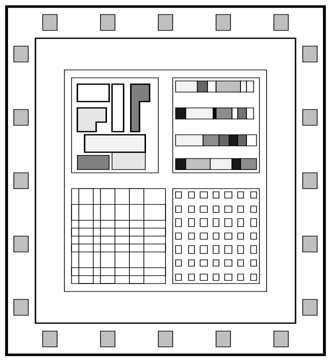
In standard-cell layout (as seen in Figure 2.5 (b)), the cells are uniform and have a rectangular shape with similar heights and different widths. The cells have fixed connections on the left and right side (clocks and/or power) that abut with each other and are placed in horizontal rows [Sher99]. The global routing phase determines where the wires switch between the rows of standard-cells. These locations are called *feedthroughs*. This design style is well-suited for moderate size circuits and medium production volumes. It provides a compromise between reasonable design time and production size because it uses a pre-designed standard cell library.

2.1.1.3 General-Cell Design

As seen in Figure 2.6 (a), the general-cell layout style is a generalization of the standard-cell layout style. The cells (available from a library or constructed as required by the design team) may be large and irregularly shaped [Sher99]. Automatic placement of general-cell designs is complicated since the cells are represented as two dimensional objects and their sizes and shapes can vary widely. Automatic routing is also more difficult (compared to standard-cells and gate-arrays) since the



(A) GENERAL CELL LAYOUT



(B) FULL CUSTOM LAYOUT DESIGN

Figure 2.6: General-Cell layout and Full-Custom layout

channels may interact in complex ways.

2.1.1.4 Full-Custom Design

This method is characterized primarily by the absence of constraints on the design process. It usually requires a hand-crafted level of automation since the lack of constraints makes synthesis tools difficult to develop [Sher99]. Full-custom design as seen in Figure 2.6 (b) is time-consuming, thus the method is inappropriate for large circuits. However, the full-custom method is widely used for smaller cells that are input to synthesis tools for high performance designs.

2.2 Graph Theory

The global routing problem is usually studied as a graph problem where either a grid graph or a gridless graph model can be used [Leng90b]. A graph is a pair of sets $G = (V, E)$, where V is a set of *vertices* and E are *edges*. A graph can be non-weighted or weighted. In a weighted graph, each edge is associated with a weight value. Figure 2.7 illustrates a graph G with vertices and edges.

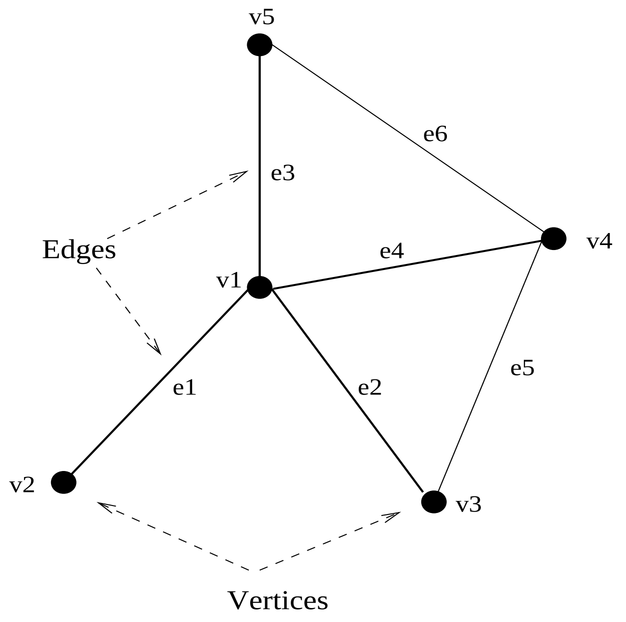


Figure 2.7: Illustration of a graph G=(V,E)

Two vertices are *adjacent* if there is a direct path, or an edge, between them. For example, vertices v_1 and v_3 are adjacent such that $e_2 = (v_1, v_3)$ is an edge. A *tree* is a connection between a subset of vertices of the graph which has no cycles. For example, edges e_1, e_2, e_3 and e_4 construct a tree which connects vertices v_1, v_2, v_3, v_4 and v_5. However, edges e_1, e_2, e_3, e_5 and e_6 are not considered as a tree due to the formation of a cyclic path.

The solution to a graph problem usually requires searching the graph by exploring all vertices. There are two important graph search techniques:

- **breadth-first search:** A graph is explored as broadly as possible by visiting a vertex, and then immediately visiting all vertices adjacent to it.

- **depth-first search:** A graph is explored as deeply as possible by visiting a vertex, and then recursively performing depth-first search on each adjacent vertex.

2.2.1 Grid Graph model

A grid graph model gives an accurate description of the routing area [Sher99], where vertices are potential positions of pins and edges are connections between pins. For standard cell layout style, each module of the circuit is represented by a vertex of the graph. If module c_i and c_j are placed adjacent to each other after the placement phase, then there is an edge connecting the vertices that represent c_i and c_j. Let $G' = (V', E')$ be the grid graph. Each edge in E' has unit capacity.

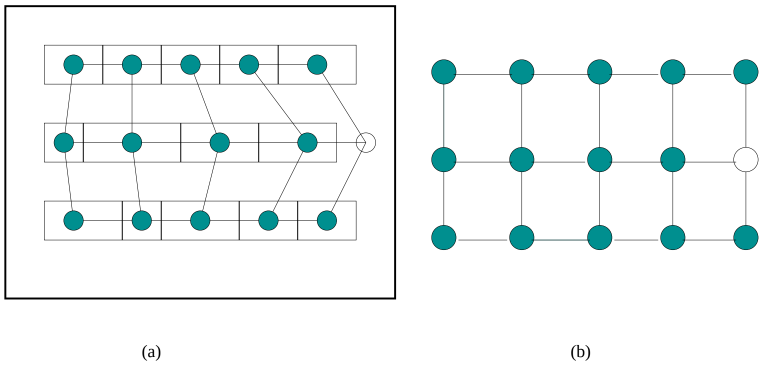


Figure 2.8: Layout of a chip with a grid-graph to represent the modules.

Figure 2.8 (a) shows the practical layout of a chip, and Figure 2.8 (b) show its corresponding grid-graph with 15 vertices. The number of vertices is determined by R_x (the row with maximum number of modules in the chip) and the number of cell rows in the chip. If a row does not have the same number of cells as R_x, empty vertices are inserted in the designated row to make the grid-graph have the same number of vertices per row.

2.2.1.1 Coarse Versus Fine Grid Graph

In a grid graph G', since every module is represented by a vertex, the total number of nets to be routed tend to increase in size. Although it accurately describes the circuit, the consequent increase in the number of constraints can result in high computational complexity. Therefore, a coarse grid graph can be constructed based on the grid graph G'. To create a coarse grid graph, G' is first cut into several subgraphs, where each forms a vertex in the coarse grid graph. Let $G = (V, E)$ represent the coarse graph. If vertices v_i and v_j are adjacent, then there is an edge between the vertices. The edge capacity of G is the number of wires that can be placed across the boundaries of two adjacent vertices. Each vertex in G is called a global cell [Hu85b].

The coarse grid graph gives a high level abstraction to the global routing problem by grouping a number of neighboring cells together. If two neighboring cells are in the same horizontal row and are connected by the same net, they are clustered together to form a global cell. The net segment that connects the two cells is also absorbed into the same global cell. A coarse grid graph tends to reduce the complexity of the global routing problem, but on the other hand decreases the precision of the graph and degrades the routing performance.

In standard cell design, the construction of a coarse grid graph is illustrated in Figure 2.9. The number of rows in the coarse grid graph is determined by the former placement technique, the vertex in the coarse grid graph or a global cell, is

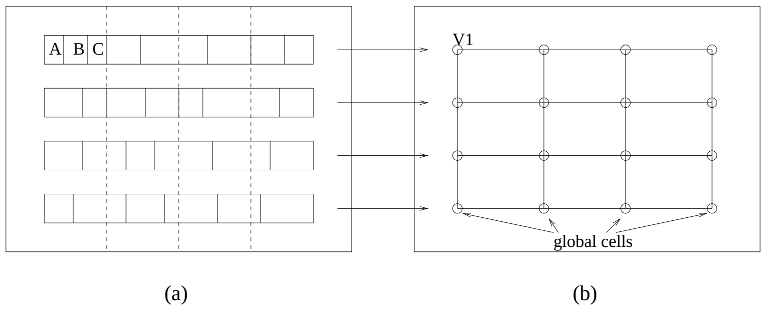


Figure 2.9: A standard cell design circuit and its coarse grid graph representation.

a collection of cells from the original circuit. For example, vertex V_1 groups cells A, B and C together. Since the row number is pre-determined, the number of columns determines the precision of the graph.

2.2.2 Connection Pattern

Global routing attempts to assign specific areas for each net within the circuit. One objective of global routing for standard cell design is to determine a connection pattern to connect all nets without violating resource constraints. The connection pattern is defined by positions for feedthroughs, pins to be connected, and channels for net segments to connect the pins.

There are several interconnection patterns to connect the pins of cells. A *Minimum Rectilinear Steiner Tree* (Figure 2.10(a)) gives the shortest route for connecting all pins. It allows wires to meet or branch at any point along its length besides the pin locations. However, computing both the optimum branching points and the resulting optimal route from the branching points to the pins increases complexity. Usually, *Minimum Spanning Tree* connections and *Chain Connections* are used as alternative interconnection topologies. In a *Minimum Spanning Tree* (Figure 2.10(b)), wires begin and end only at the pin locations. The construction of a *Minimum Spanning Tree* will be discussed in section 2.2.3. In the *Chain Connections* (Figure 2.10(c)) method, there are no branches, since each pin is connected to the next in the form of a chain. These two connections are easy to implement, but the trade off is that the interconnects through all the pins are longer. *Source-to-sink Connections* (Figure 2.10(d)) consider the critical path length between the source and sink pins and are used in timing-driven global routing.

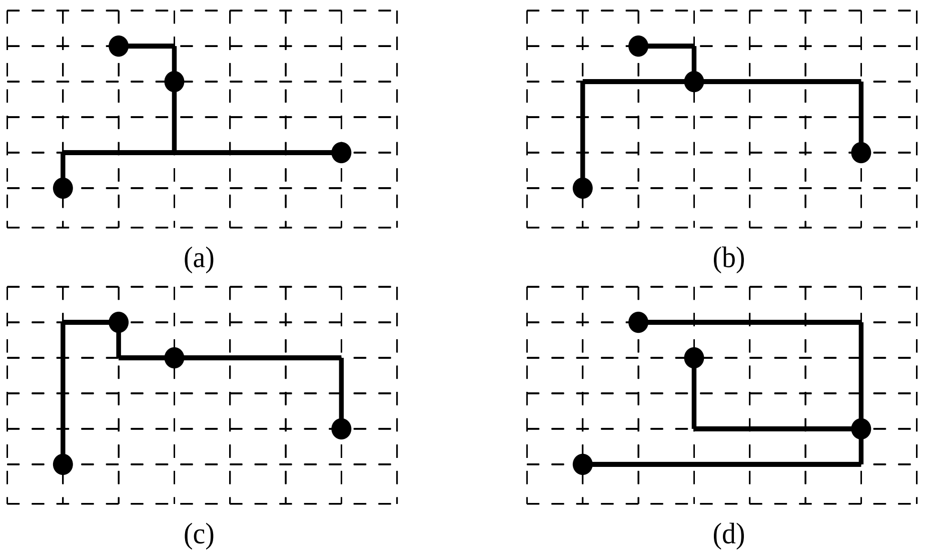


Figure 2.10: (a) Minimum Rectilinear Steiner Tree. (b) Minimum Spanning Tree. (c) Chain Connections. (d) Source-to-Sink Connections

A cost function is used to measure and compare the interconnections by calculating the *estimated wire length.* Thus the objective of a global router is to choose the optimal connection wiring patterns that minimize the overall cost function. An efficient and widely used method to estimate the wire length is the Half Perimeter Wire Length (HPWL) method, in which the wire length is approximated by half of the smallest bounding rectangle enclosing all the pins, as shown in Figure 2.11. For all two-terminal and three-terminal nets this method gives accurate wire length, provided that the routing does not go beyond the bounding rectangle. For the nets which have more than three terminals, the yielded wire length underestimates the actual wire length. The HPWL method can also provide the best estimation for the Steiner tree wiring scheme. In practical circuits, two-terminal and three-terminal nets constitute over 98% of the nets in a circuit, thus using the HPWL method is considered to be a good estimation technique [Shah91].

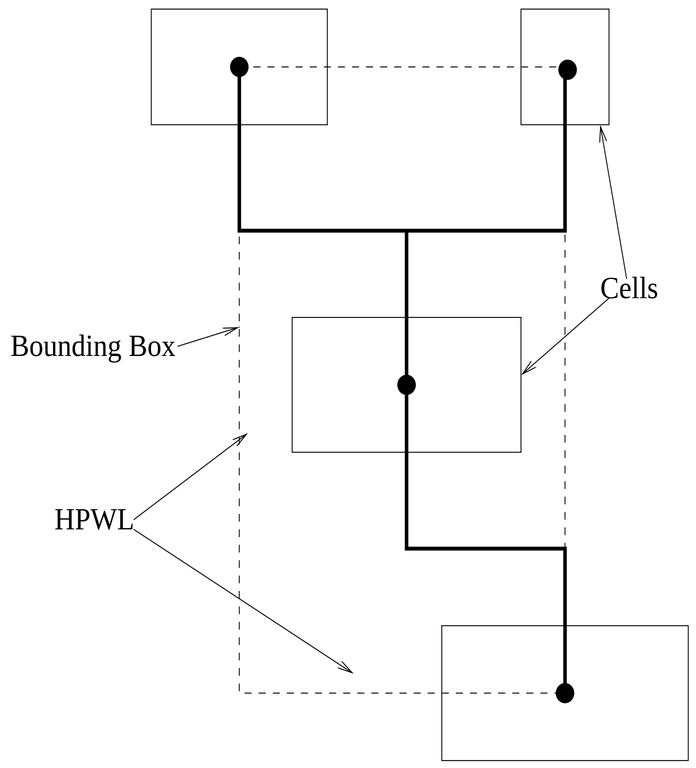


Figure 2.11: Approximation wire estimation and congestion

2.2.3 Minimum Spanning Tree (MST)

The Minimum Spanning Tree problem is formulated as follows: given a graph $G = (V, E)$, select a subset of edges $E' \subseteq E$, such that E' constructs a tree which connects all the vertices V and the total cost of edges $\sum_{e_i \in E'} w(e_i)$ is the minimum cost, where $w(e_i)$ is the cost of the edge e_i.

```
Kruskal's Algorithm
1. Start;
2. Order all edges in increasing weight;
3. Start with $N$ sets, where each set $T_i$
   represents a node ;
4. **While** (there are edges connecting different sets)
       Choose the shortest edge $E_{ij}$
         which connects two different sets $T_i$ and $T_j$;
       Merge $T_i$, $T_j$ and $E_{ij}$ into set $T_i$;
   **End While**
5. End;
```

Figure 2.12: A Minimum Spanning Tree Algorithm

Kruskal's algorithm [THC90] is one of several techniques used to construct Minimum Spanning Trees. Its pseudo-code is illustrated in Figure 2.12. For a given graph $G = (V, E)$, Kruskal's algorithm first orders all edges in increasing weight and assigns each vertex to a set T_i. Each set represents a partial spanning tree. Next, it chooses the remaining smallest weight edge $E(ij)$, where i and j are from different sets T_i and T_j. By merging the two sets together a new set is created. This selection continues until there are no remaining edges to be added to the spanning tree. The complexity of Kruskal's algorithm is $O(E \lg E)$ [THC90], where E is the number of edges.

Figure 2.13 gives an example of constructing an MST using Kruskal's algorithm.

There are 6 sets in the beginning:$\{A\}$, $\{B\}$ $\{C\}$ $\{D\}$ $\{E\}$ and $\{F\}$. First, edge $E(de)$ is selected from the ordered edge list, the two sets$\{D\}$ and $\{E\}$ are merged into a new set $\{D, E\}$. Next, edge $E(ab)$ is selected, the two sets $\{A\}$ and $\{B\}$ are merged into a new set $\{A, B\}$. As the same edge selection process continues, edge $E(ec)$, edge $E(ef)$ and edge $E(ac)$ are subsequently selected. The final MST with minimum cost is found (as seen in Figure 2.13 (f)) using five edges with an optimal cost of 18 units.

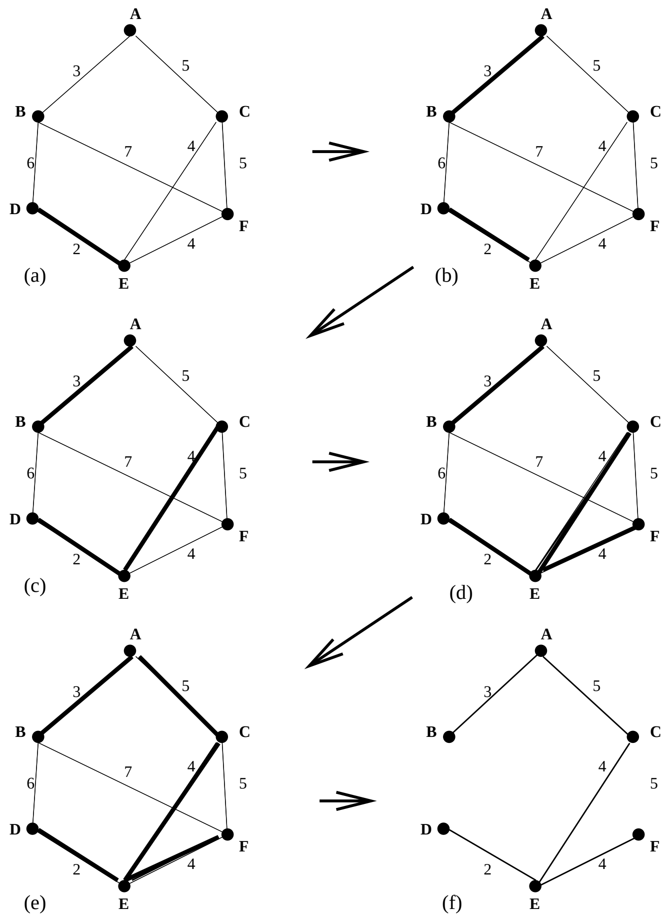


Figure 2.13: An example of MST construction

2.2.4 Netlist and Admissible Routes

A netlist provides information about the connectivity of the circuit. It shows the wiring requirement of a circuit and consists of a set of nets. Each net is identified by a name and a list of pins or terminals (as seen in Figure 2.14). The name usually specifies the type of the wire as being either global or local. Figure 2.14 shows a netlist in a standard cell layout which consists of two nets.

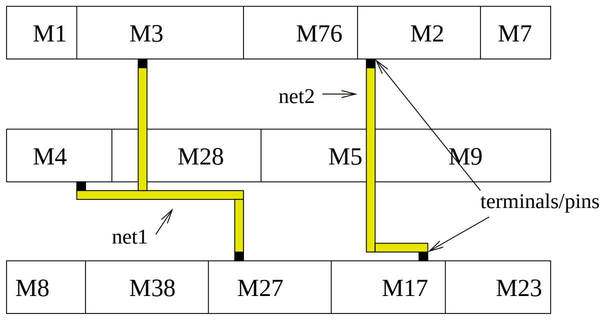


Figure 2.14: A netlist consists of two nets.

An admissible route is a path that a net can take in the final layout [Behj02]. For any net, there are several admissible routes. Theses routes can be defined by a certain connection pattern such as a Minimum Spanning Tree (MST) or a Minimum Rectilinear Steiner Tree (MRST). However, for a net with a large number of terminals, it is not feasible to consider all possible routes. Usually a subset of all routes can be categorized as admissible routes.

2.3 Global Routing for Standard Cell Design

As explained earlier, standard cell design consists of a set of logic library cells arranged in rows. The space between two rows is referred to as a channel which is an open-ended region with pins on the top and bottom boundaries. Routing for standard cell design is normally performed within channel areas.

There are two objectives of global routing for standard cell design. The first is to determine a net connection pattern without violating resource constraints, as discussed in section 2.2.2. The second objective is to minimize channel density. Total density is normally used to measure routing area. The local density of a channel at a given x-coordinate is the number of nets that intersect a vertical line segment which passes through that channel at the coordinate [Sher99]. The channel density is the maximum local density over all x. The total density of a standard cell layout is the sum of channel densities over all channels. The area required by the chip is a function of the longest cell row and the total channel density. Therefore, minimizing the total channel density is considered as an important objective of global routing.

For example the chip presented in in Figure 2.15 has four rows of cells and five channels. The channel densities for channels two, three and four are 7, and for channels one and five are 5 and 6 respectively, producing a total channel density of 32 (7+7+7+5+6=32).

The standard-cell design style does not impose any restriction on channel heights and therefore this guarantees 100% routability. When needed, feedthroughs can be inserted into the cell row to provide vertical interchannel connection. Insert-

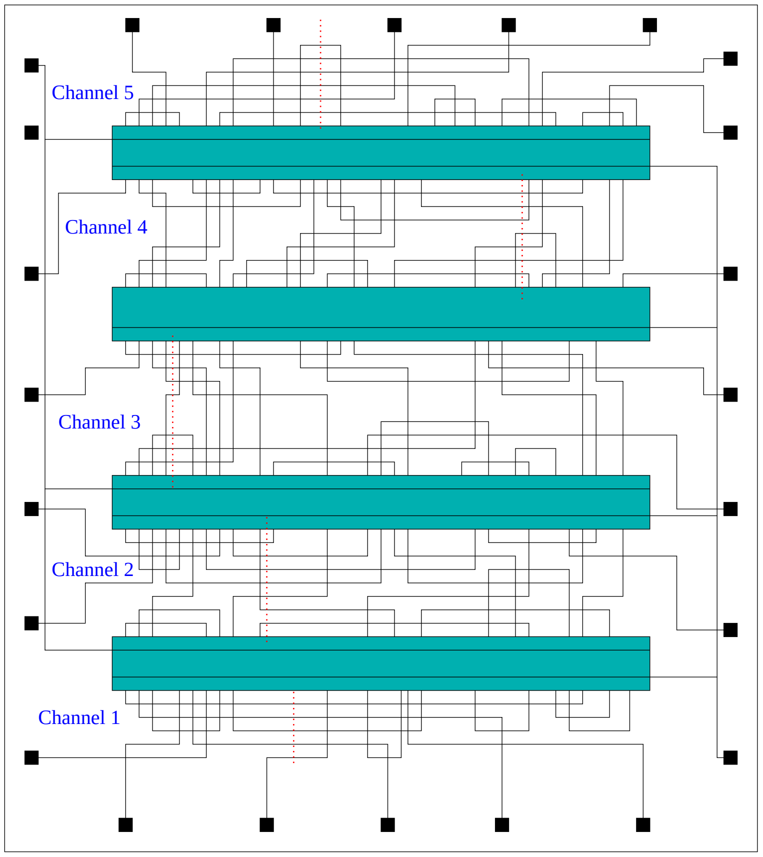


Figure 2.15: Channel density and total density.

ing feedthroughs to cell rows except the longest row does not change the width of the chip. In a typical two layer standard cell design, one layer is used for horizontal channel segments, and the second is used for vertical connections between the cell and the channel segments in the same channel. These vertical tracks are called feedthroughs which accomplish the interchannel connections. Normally, feedthroughs are realized in two ways:

- By making use of the built-in **feedthrough wires** (Figure 2.16(a)) that may be available within a standard cell. Feedthrough wires cross the cell without any relationship to the functional circuitry [JRos88]. The pair of pins on the ends of a feedthrough wire are called equivalent pins.

- By making use of **feedthrough cells**(Figure 2.16(b)) which are inserted between functional cells. Feedthrough cells are standard cells that contain no logic and only vertical feedthrough wire(s) [JRos88].

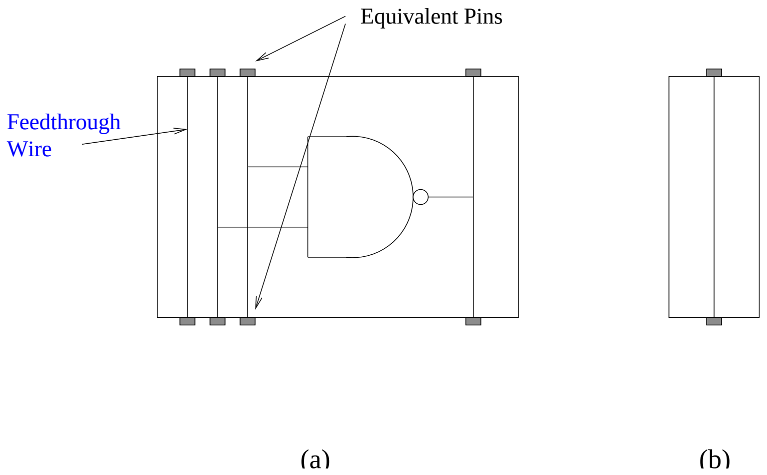


Figure 2.16: (a) A two input NAND gate standard cell with a feedthrough wire. (b)A feedthrough cell.

Figure 2.17 shows two nets that span multiple rows. Net_1 needs two feedthrough cells inserted to complete the connection. Net_2 uses the built-in feedthroughs such that no more feedthrough cells are inserted. The cells that Net_2 passes through in the second and third row have equivalent pins on the top and bottom of the cells.

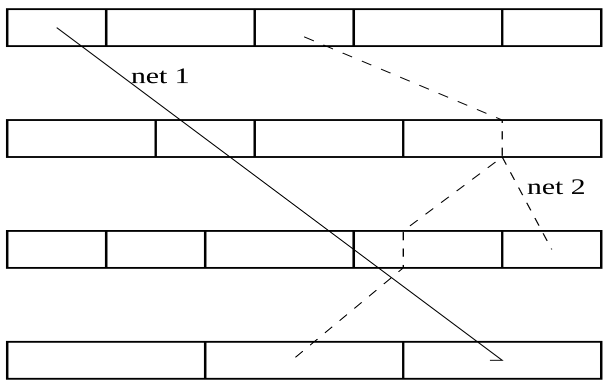


Figure 2.17: Example of two nets that span multiple rows.

2.4 Parallel Programming

High performance computers are increasingly in demand by engineering discipline in general and VLSI CAD in particular. Achieving high performance depends not only on using faster hardware but also on major improvements in computer architecture and processing techniques. Research and development of multiprocessor systems are aimed at improving throughput, reliability, flexibility and availability.

Conventional sequential algorithm based CAD will finally reach its limitation in the future. It is becoming a necessity to develop parallel algorithms to solve complex combinatorial optimization problems. Parallel algorithms vary in performance and depend heavily on the architecture of the parallel computer to be used.

2.4.1 Classification

Parallel systems can be classified into SIMD and MIMD architectures by instruction delivery mechanism and data stream. SIMD stands for single-instruction, multiple-data and MIMD stands for multiple-instruction, multiple-data.

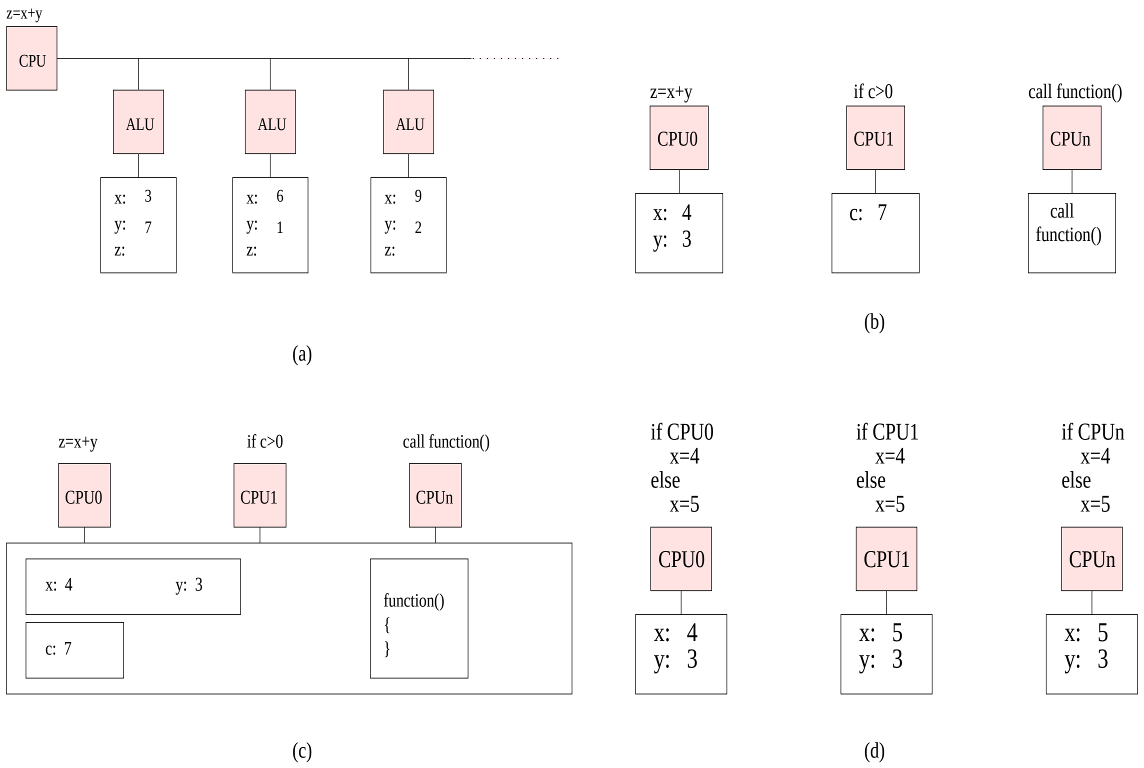


Figure 2.18: (a) Single Instruction Multiple Data (SIMD) (b) Distributed Memory Multiple Instruction Multiple Data (MIMD) (c) Shared Memory Multiple Instruction Multiple Data (MIMD) (d) Single Program Multiple Data (SPMD)

In a SIMD (Figure 2.18 (a)) parallel processor, a central controller broadcasts the same instruction to different processors, and each processor executes the instruction on its own data. In an MIMD multiprocessor, processors execute different instructions on different data. MIMD can also be further classified into two models by the memory organization: distributed memory MIMD(Figure 2.18 (b)) and shared memory MIMD (Figure 2.18 (c)). The memory organization is discussed in the following section. Single program Multiple Data (SPMD) (Figure 2.18 (b)) is a special case of MIMD. In this parallel model, the same program is executed on several processors according to different conditions.

2.4.2 Shared Memory and Distributed Memory

MIMD multiprocessors are classified as either being shared memory or distributed memory architectures [Bane94]. In a shared memory computer, processors can have access to all memory locations, and therefore communications and synchronization are low. However, no more than one processors can access the same data at the same time. In a distributed memory MIMD computer, each processor has its own local memory and works without considering to be blocked by other processors. Communication is largely increased when each processor tends to broadcast messages informing other processors of current changes.

2.4.3 Amdahl's Law and Parallel Speedup

The most commonly used metric to measure parallel processing performance gain is parallel speedup [Sun90]. Speedup has two different definitions. The first referred to as absolute speedup, focuses on speedup achieved to solve a problem on N processors. Absolute speedup compares the best sequential algorithm with the parallel algorithm under consideration. On the other hand, relative speedup considers inherent parallelism of the parallel algorithm. It is the ratio of elapsed time of the parallel algorithm on one processor to the elapsed time of the parallel algorithm on N processors.

The relative speedup has been used more commonly to evaluate the parallel performance gain and its formulation is given by $Amdahl's\ law$. Ideally, an algorithm distributed among N processors would complete in $1/N$ time, leading to an N times increase in parallel performance. However, any given parallelized algorithm will contain a serial portion. These portions can not run any faster on a number of processors than on a single processor. Let $T(N)$ be the time required to complete the task on N processors, the relative speedup can then be defined as:

$$S(N) = \frac{T(1)}{T(N)} \tag{2.1}$$

In many cases, since the time $T(1)$ has both a serial portion T_s and a parallelized portion T_p, the speedup can be further defined as:

$$S(N) = \frac{T(1)}{T(N)} = \frac{T_s + T_p}{T_s + T_p/N} \tag{2.2}$$

According to the equations given above, The parallel time can not be decreased by a factor of $1/N$, since the serial time remains constant no matter how many processors a programmer utilizes to run the algorithm.

2.5 MCNC Benchmarks

The global routing algorithms implemented in this thesis are tested on the MCNC benchmarks [Kozm91]. These benchmarks are widely used to verify the effectiveness of algorithms for VLSI physical design subproblems such as partitioning, placement and routing.

Circuit	# nets	# cells	# pads	# pins
fract	147	125	24	876
struct	1920	1888	64	10814
prim1	1133	752	81	5614
ind1	2478	2271	814	19186
prim2	3817	3014	107	22371
bio	5742	6417	97	41886
ind2	13419	12142	495	95818
ind3	21938	15059	374	136084
avq.small	22124	21854	64	152334
avq.large	25384	25114	64	165374

Table 2.1: MCNC benchmarks

Table 2.1 shows the statistics of MCNC benchmarks, which consists of test circuits ranging in size from 147 cells to 25,000 cells. It lists the number of nets, cells, pads and pins for each circuit. Table 2.2 lists the number of rows, chip width and chip height for each circuit.

Circuit	No. rows	Width (Microns)	Height (Microns)
fract	6	828	754
struct	21	2404	2610
prim1	16	5231	5250
ind1	15	3085	3672
prim2	28	11354	7350
bio	46	5339	5162
ind2	72	14998	14456
ind3	54	28169	26880
avq.s	80	9343	9338
avq.l	86	9502	9918

Table 2.2: The chip dimension of MCNC benchmark circuits

The circuits are grouped into three classes according to their sizes: small, medium and large, the small size group includes *fract*, *struct* and *prim*1 circuits, the medium group includes *ind*1, *prim*2 and *bio* circuits, and the large one consists of *ind*2, *ind*3, *avq.small*, *avq.large* circuits.

2.6 Summary

Due to the high complexity of the VLSI design process, it is broken down into several more manageable smaller tasks. One of these sub-tasks is physical design, which is still considered complex. As a result, this complexity is handled by dividing the physical design problem into more tractable sub-tasks and global routing is one of these sub-tasks. Physical design automation highly depends on the layout style used. Different layout styles, such as full-custom, gate-array, and standard-cell layouts can achieve different trade-offs among speed, cost, fabrication time, and degree of design automation. Background material on global routing phase, parallel programming techniques and test benchmarks were also introduced in this chapter. Chapter 3 will briefly review previous work on the global routing problem.

Chapter 3

Literature Review

This chapter briefly reviews previous work and research on the VLSI global routing problem. There are mainly two distinct methodologies for global routing: sequential techniques and concurrent techniques (as illustrated in Figure 3.1). Sequential techniques tend to route nets according to certain sequences, and are further classified into two-terminal and multi-terminal algorithms. Two-terminal algorithms include maze router algorithms [Lee61], line-probe router algorithms [KMik68] and shortest path based algorithms [Sher99]. Multi-terminal algorithms are mainly based on Steiner tree techniques. Concurrent techniques on the other hand route all nets concurrently, and are classified to belong to either Integer Programming based algorithms or hierarchical based algorithms.

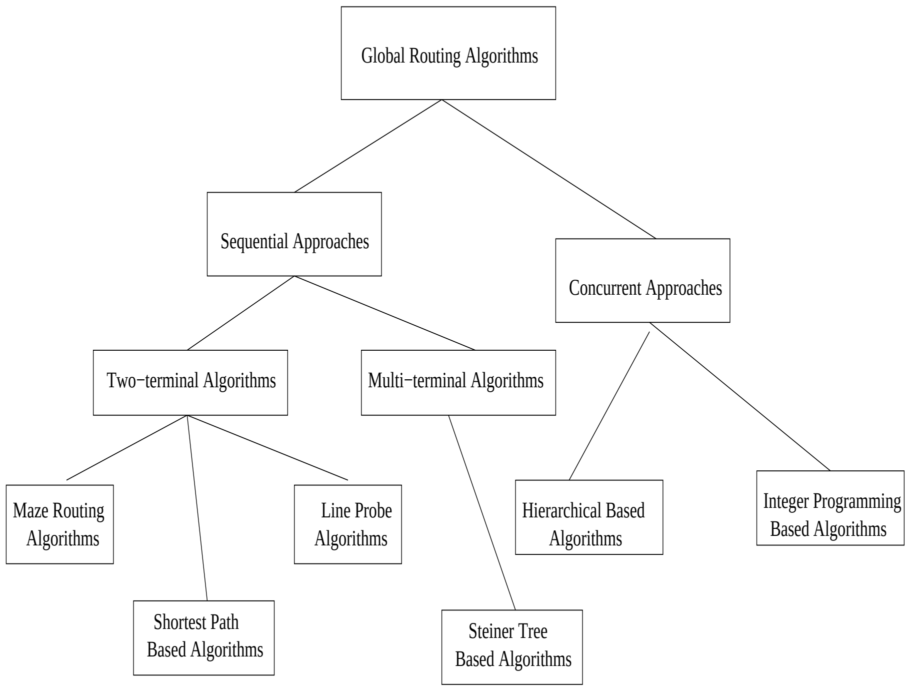

Global Routing Algorithms
Sequential Approaches
Concurrent Approaches
Two−terminal Algorithms
Multi−terminal Algorithms
Maze Routing Algorithms
Line Probe Algorithms
Shortest Path Based Algorithms
Steiner Tree Based Algorithms
Hierarchical Based Algorithms
Integer Programming Based Algorithms

Figure 3.1: Methodologies for global routing.

3.1 Sequential Approaches

Sequential routing algorithms [Sher99], as the name implies, route nets in a sequential manner by ordering nets according to their priority. The quality of a sequential global router largely depends on the ordering of nets. Once a net is routed, it occupies resources in the form of tracks and feedthroughs. Therefore nets routed later can not use the occupied resources. Determining a good net order prior to the routing process is a very important task for sequential global routers. Usually, the nets are sequenced according to their criticality, perimeter of the bounding rectangle and number of terminals [Sher99]. Since nets with high priority often play key roles in determining the performance of the circuit, following the net order determination, these nets are routed first. A brief review of sequential approaches will be introduced next.

3.1.1 Two-terminal Sequential Algorithms

Maze routing algorithms are the most widely used for routing two-terminal nets [Moor59]. Lee [Lee61] introduced the first grid expansion maze routing algorithm for a two terminal net on a grid graph. This routing heuristic uses a breadth-first-search method (BFS) to explore the grid graph. In order to find the shortest path between a source point "S" and a target "T", the maze global router first constructs several initial paths at "S" and then expands them until one path reaches "T" (as seen in Figure 3.2).

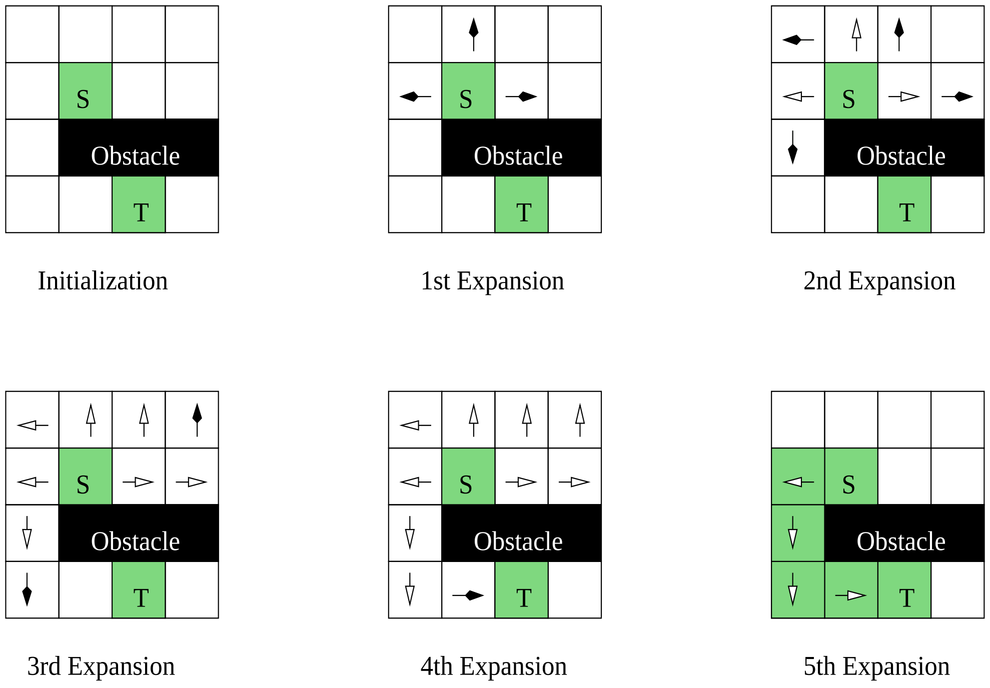


Figure 3.2: Maze routing algorithm

When the target is reached, a retrace phase begins to retract all vertices which identify the path. Maze routing algorithms are simple and can guarantee finding the shortest routes between any two pins if such route exists [Souk78]. However, there are some major drawbacks of sequential maze routing algorithms. Firstly, they tend to pick a shorter path with many bends and vias, instead of a longer path with fewer bends. Vias occupy valuable routing space so they enlarge the whole chip area and cause delays [Sher99]. In the worst case, maze routing might require to explore all vertices in the grid graph to find a shortest path which has space complexity $O(n^2)$, where n is the number of vertices. Furthermore, maze routing heuristics usually require a large amount of memory and their performance degrades rapidly when the size of the grid increases in size[Sher99].

To overcome the drawbacks of maze routing algorithms, line-probe routing algorithms [KMik68] were developed. Line-probe routing algorithms perform instead a bidirectional search. Straight search lines are extended from both source point "S" and target "T" in all four directions, as illustrated in Figure 3.3. If a search line extended from source meets a line extended from target, then a path is constructed. In some cases, these trial lines may hit some obstacles. To escape obstacles, new directions are extended for some base points on the trial lines. The space complexity of line-probe algorithm is $O(L)$, where L is the number of line segments. Obviously, line-probe algorithms are faster than maze routing algorithms by reducing the space complexity [Sher99]. However, they may not produce the shortest path between two points since they are greedy in nature.

Shortest Path Based approaches use Dikjstra's Algorithm [Dijk59] to route two-terminal nets. Given a routing graph $G = (V, E)$, Dikjstra's Algorithm can find the

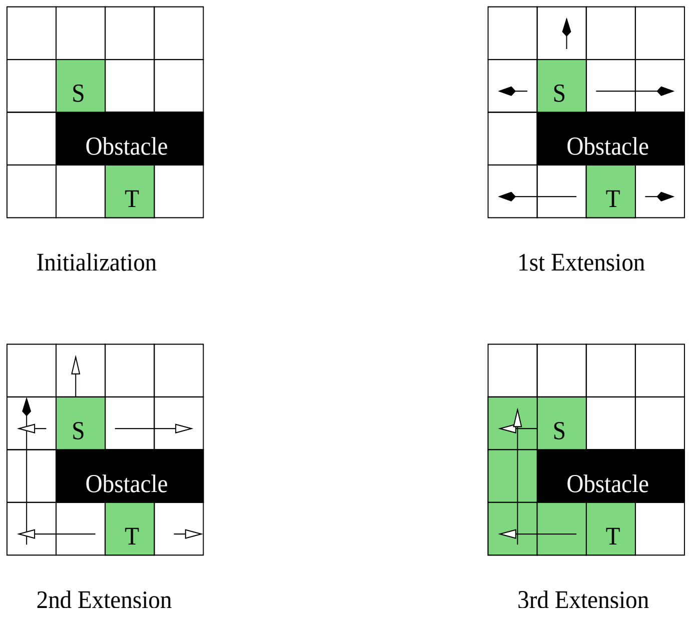


Figure 3.3: Line probe routing algorithm

shortest path in G joining a source point "S" and a target point "T", where $S \in V$ and $T \in V$. When evaluating the length of an edge in the path, the algorithm uses a factor α ($\alpha > 1$) that tends to increase the length of a congested edge in order to reduce edge congestions.

The maze routing and line-probe algorithms are not originally designed to route multi-terminal nets, however, several approaches have been proposed to extend them to solve multi-terminal routing problems [Sech88]. In these approaches, the multi-terminal nets are decomposed into several two-terminal segments, and each segment is routed by using either a maze router or a line-probe algorithm.

3.1.2 Multi-terminal Algorithms

For a multi-terminal net, finding its shortest connection is usually achieved by constructing *Minimum Spanning Trees* (MST) or *Minimum Rectilinear Steiner Trees* (MRST), which were explained in section 2.2. The *Minimum Spanning Tree* is the shortest path to connect the nets with each edge only starting or ending at pin locations. The *Minimum Steiner Tree* problem can be stated as: given a set P of n points, find a set S of *Steiner points* such that the Minimum Spanning Tree (MST) over $P \cup S$ has minimum cost.

Hanan [Hana85] showed that all possible Steiner points must lie within the bounding box of the outer most nodes of a net. He also showed that any *Minimum Rectilinear Steiner Tree* (MRST) can be found over the subgrid which is formed by the intersection of the vertical and horizontal lines drawn at each node in the net. These intersecting points are called Hanan points. Figure 3.4 shows a point set in solid dots and the corresponding Hanan grid and points. Figure 3.5 illustrates the existence of MRSTs within Hanan points.

In TimberWolf 3.2 [Sech86], the global router tends to perform the optimization in two steps: in the first stage, every net is connected by a Minimum Spanning Tree based on wire length. This is followed by an iterative algorithm (simulated annealing with temperature = 0) to further improve the assignment of net segments to channels. The goal of the global router is to connect all nets while minimizing the total number of wiring tracks. As a typical sequential algorithm, the net order problem is one of the main disadvantages of such a technique. Moreover, the global router in Timbewolf 3.2 is not capable of predicting congested areas in channels

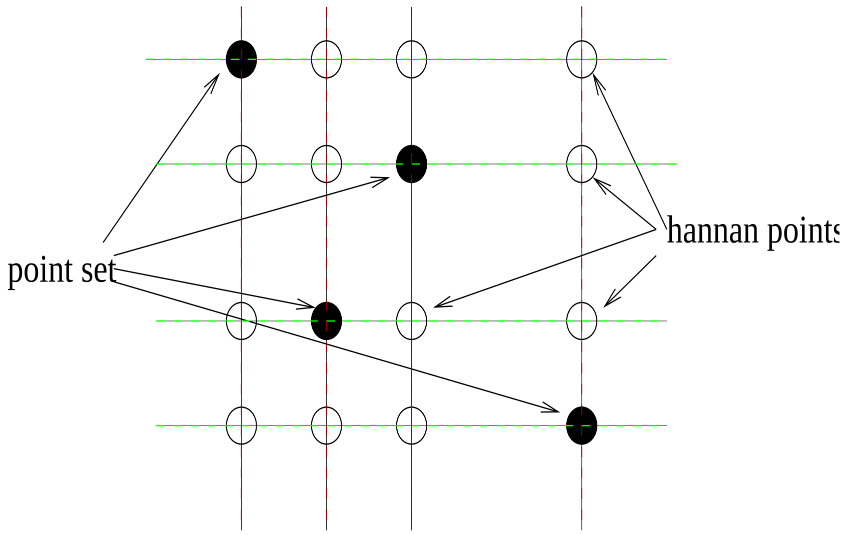


Figure 3.4: The Hanan grid and the set of Hanan points.

when the net segments are added.

In [KW88], K. Lee improved the TimberWolf global router by constructing efficient Steiner trees. The router considers the features of the row based layout placement, such as equivalent pins and built-in feedthroughs, and generates the L-switchable segments to construct the Steiner tree. The new global router improves performance by using a more accurate interconnection model. Thus the total channel density and congestion are reduced.

In [Sarr89], a global router was proposed to find the Steiner min-max trees. This is a two-step algorithm where the first step involves finding a net order for all nets. Each net is given a net number according to a function of bounding length, priority and multiplicity. In the second step, an optimal Steiner min-max tree is found for each net. The Steiner tree whose maximum weight edge is minimized can contribute to minimizes traffic in the densest channel.

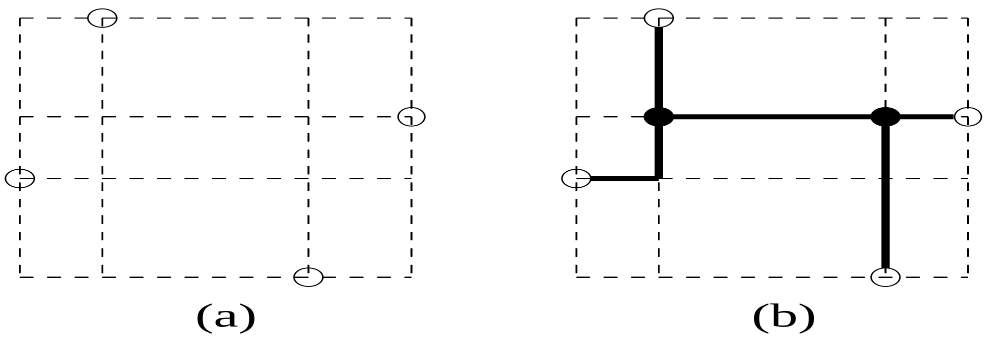


Figure 3.5: There always exits an MRST with Hanan points chose from the intersection of all the horizontal and vertical lines through all the points

S. Miriyala [SM91] proposed a heuristic algorithm which produces optimal solutions for the Steiner tree problem in presence of obstacles. There are two steps in this algorithm. Initially, an upper bound is produced if the terminals are on the boundary of rectangle. Following that, a rectilinear steiner tree is found within the bound produced by the previous step.

In [Swar93], a row-based global router is proposed for standard cell circuits. Its main feature is the dynamic modification of Steiner trees produced. This row-based global router explicitly minimizes chip area, and also takes timing constraints into account.

Some non-rectilinear approaches [SB91] have also been proposed that can reduce the cost of the optimal trees by 10%-12%. Non-Rectilinear Steiner tree based algorithm uses a non-rectilinear geometry. An example of the derivation of a Steiner tree for a simple two-terminal net in a 4-geometry is shown in Figure 3.6 (b), while Figure 3.6 (a) shows the derivation of a Steiner tree for the same net in rectilinear geometry. Clearly, the tree length in 4-geometry is shorter than the one in the rectilinear geometry.

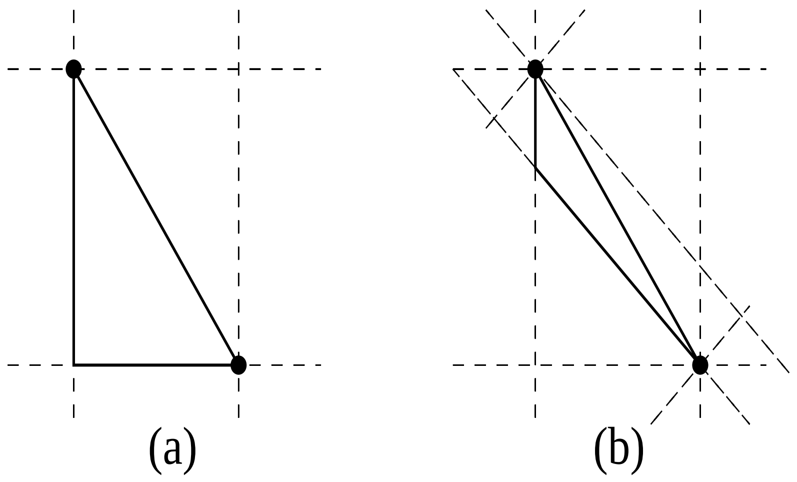


Figure 3.6: (a) 2-geometry. (b) 4-geometry.

Other previous research focused on pre-processing techniques that tend to estimate the wire length to shorten the running time of the following stages. Research in [Pedr89] proposed an interconnection length estimation model for standard cell layouts. The prediction of interconnection helps to determine the routability of the proposed logic design or to evaluate the quality of placement and global routing algorithms.

In [Cho00], Jun Dong C. proposed a wiring space and length estimation approach. It divides the routing region into a top-down quad-tree hierarchy. A lower-bound density of cells and an upper-bound are obtained for the worst case of cells. Assume d_0 is the estimated lower density in a $m \times m$ two-dimensional array and $\alpha + \beta = 1$, the total wire length is $(2\alpha + \beta)4m^2 d_0/3$. Where α is the percentage of diagonal combinations and β is the percentage of adjacent combinations of nets in a quadrisection map.

Furthermore, in order to obtain better solutions for sequential approaches, some improvement heuristics are developed to improve congestion and channel densities. These approaches are developed mainly for gate arrays; however, it provides a frame work that can be adopted for standard cell design. Many recent global routers are two phase heuristics: a coarse routing phase followed by a refinement phase. One such routing technique is developed by Ting and Tien [Ting83] for gate arrays. The basic idea of the heuristic is as follows:

- Route every net as if it were the first net to be routed; i.e. pay no attention to boundary overflow conditions;

- After all nets have been routed, identify the boundaries that are overflowed and the amount of overflow;

- Identify the nets that use these overflow boundaries and form a bipartite graph with one part of the vertices representing the nets and the other part of the vertices representing the overflowed boundaries. An edge connects a net vertex to a boundary vertex if the net uses that boundary. The bipartite graph shows the supply-demand situation among boundaries and nets. A subset of the nets are selected for rerouting in a "greedy" manner.

3.2 Concurrent Approaches

Concurrent approaches can avoid the net-ordering problem which degrades the performance of sequential approaches by routing all nets simultaneously. Furthermore, concurrent approaches have a global view of the chip area therefore they can predict congestion. Concurrent approaches are classified into hierarchical based algorithms and Integer Programming (IP) based algorithms.

3.2.1 Hierarchical Based Approaches

Hierarchical approaches partition the circuit area into sub-regions which are called global cells. Global routing problems are solved for each global cells and then the solutions are combined together.

A hierarchical based routing approach was first proposed by Burstein and Pelavin [Burs83], and was enhanced by Heisterman and Lengauer [Heis91]. In this approach, the whole routing area can be represented by a cut tree. As seen in Figure 3.7, the root of the cut tree is the entire chip and its children are bipartitions of the chip. The grand children of the root are bipartitions of its children, and so on. Let the entire chip be arranged in an array of cells. A cell is a module placed onto the silicon wafer. At each level of the hierarchy, a node is divided into a 2x2 supercell. Each net is represented by a set of cells which is called terminal cells. If a supercell is a terminal cell for net k, then at least one pin of the net is located somewhere in this supercell. Each supercell contains a maximum of one pin per net, thus if a net lies within a supercell, it is not routed at that level.

The hierarchical routing heuristic starts at the root of the tree and visits each

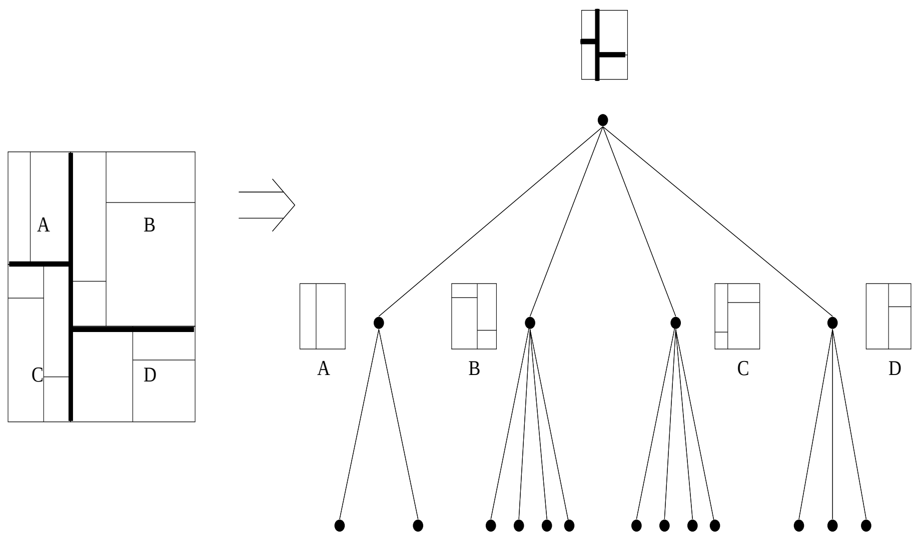


Figure 3.7: A floorplan and its preprocessed cut-tree

node until it reaches the leafs of the tree, i.e. the bottom of the hierarchy. At each node, a 2x2 grid routing problem is solved by using integer programming. Before attempting the next level, the solutions for all nodes in this level of the cut-tree are combined. The resulting routing influences the definition of the routing problems for the nodes in the next lower level.

In [Brou90] a global router called PHIGURE hierarchically decomposes the whole process into separate independent tasks. These sub-tasks are suitable for parallel execution and adaptive simplex solution to add feedthroughs and adjust channel heights for row-based layout. The main disadvantage of this approach is that it only targets specific parallel architectures in the form of shared memory multiprocessors. In addition, congestions are ignored during the feedthrough assignment.

The hierarchical approach is further investigated in [Heis91]. The global problem is decomposed into small integer problems. These problems are classified into different cases where some simple routable nets are prerouted. This preprocessing method reduces the whole problem size thus yields a reduction of the processing time.

An alternative hierarchical approach called 2x2 cell routing was proposed by Marek Sadowska [Kuh86]. The hierarchical global router uses 2x2 cells to enumerate all possible types of nets as well as connections that may occur. In Fig. 3.8 a complete tabulation of nets and connections is introduced with six types of 2-terminal nets, four types of 3-terminal nets and one type of 4-terminal nets. Each type of a k-terminal net represents the possible pin configuration in the 2x2 supercells. For each net type, there are various possible connections. Let k(i) be the number of nets and $p(i,j)$ denote the j^{th} possibility to route a net of type i. The global routing problem can be expressed as follows: for $i = 1, 2, \ldots, 11$, find the number of nets of which type i can be realized in each possibility to complete the routing. Let $x(i,j)$ denote the number of nets of type i using the j^th possibility. The values of $x(i,j)$ can be determined by solving the integer programming problem.

Let X_i be the number of overflowed nets of type i.

Let V_1, H_1, V_2, and H_2 be boundaries.

Let v_1, h_1, v_2, and h_2 denote channel capacities at the V_1, H_1, V_2, and H_2 boundaries respectively.

$\widetilde{H_1}$ = {(i,j) | p(i,j) crosses vertical boundary H_1}.

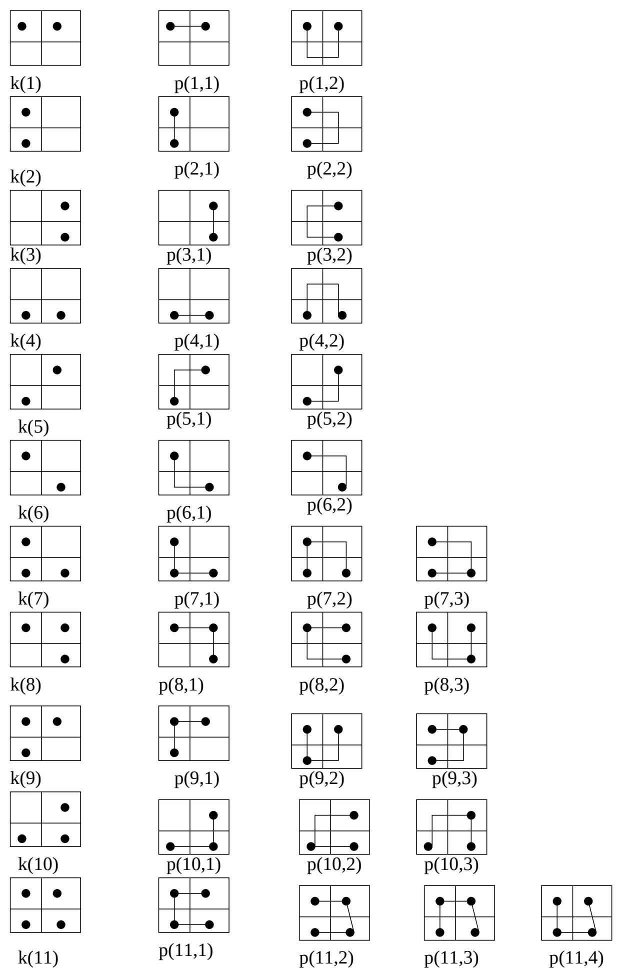


Figure 3.8: Pin configuration and net types in 2x2 cell array.

$\widetilde{H}_2$, $\widetilde{V}_1$, and $\widetilde{V}_2$ are defined similarly.

y_1, y_2, y_3 and y_4 denote the number of free capacities on the V_1, H_1, V_2 and H_2 boundaries, respectively.

The integer program is formulated as following:

$$\min \sum_{i=1}^{11} X_i \tag{3.1}$$

$$\max \sum_{i=1}^{4} y_i \tag{3.2}$$

Subject to

$$X_i + \sum_j x(i,j) = k(i) \qquad \forall i = 1, 2, \ldots, 11. \tag{3.3}$$

$$\sum_{(i,j)\in\widetilde{V}_1} x(i,j) + y_1 = v_1 \tag{3.4}$$

$$\sum_{(i,j)\in\widetilde{H}_1} x(i,j) + y_2 = h_1 \tag{3.5}$$

$$\sum_{(i,j)\in\widetilde{V}_2} x(i,j) + y_3 = v_2 \tag{3.6}$$

$$\sum_{(i,j)\in\widetilde{H}_2} x(i,j) + y_4 = h_2 \tag{3.7}$$

Equation 3.3 is a necessary condition for a feasible routing solution. Equations 3.4, 3.5, 3.6 and 3.7 are the capacity constraints respectively. This model is small enough that it can be solved by using general integer programming techniques but at the expense of large CPU time.

3.2.2 Integer Programming Based Approaches

Integer Programming approaches formulate the global routing problem as a 0/1 Integer Programming (IP) problem such that the objective is to connect each net simultaneously without violating the constraints.

Hu and Shing [Hu85b] first proposed a Linear Programming (LP) formulation of the global routing problem. They used column generation techniques to produce potential candidates. This approach has two limitations: the problem size was very large and thus using the Simplex method to solve the LP was slow.

R.M. Karp [RK87] developed an algorithm to solve the Integer Linear Programming (ILP) formulated global routing problem using a randomized rounding technique. At first, an initial solution is found with the use of a linear program solver. Next, by rounding any fractional numbers in the solution to 0 or 1 using a randomized technique the final solution was obtained. The main drawback of this approach is that it can not handle multiple pin nets correctly, although it finds the global routing solutions of all nets simultaneously.

A. Vannelli [Vann91] extended Hu and Shing's Linear Programming (LP) model by reducing the complexity of the global routing problem. In this work, a new Linear Programming (LP) solution technique called interior point method [**?**] was introduced to solve the problem efficiently. Since interior point methods explore the inner spheres of the LP polytope (linear constraints), it is faster than the standard simplex-based algorithm. To further reduce the problem size and speed up the running time, only minimal rectilinear spanning trees or near minimal rectilinear spanning trees are selected.

In [Behj02], a global routing approach that combines wire length and congestion estimation was proposed. First a set of MST is constructed for each net. Next, an Integer Linear Programming (ILP) model is formulated with the objective of minimizing the wire length. Congestion estimation is incorporated in the objective function. Finally, the ILP model is relaxed as a Linear Programming (LP) model which is solved using Interior Point algorithms. A rounding algorithm was developed to turn the fractional solutions of the LP problem to integer solutions.

Related to linear programming (LP) approaches, global routing problems can be considered as an extension of network flow graph models [Baza77]. Vertices in these graph models are the pins to be connected, and each edge consists of a capacity equal to the number of nets to be routed through the corresponding channel. After all flows are generated, the network flow algorithm is executed to determine the number of nets that may be distributed through a channel.

A single-commodity flow formulation for global routing problem was first proposed by Meixner and Lauther [Meix90]. This global router combines network flow with linear assignment technique. First, the network flow is obtained by a steiner tree heuristic based algorithm and wires are distributed to channels concurrently. Next, feedthroughs and bridges are assigned to associated nets through linear assignment. This approach not only reduces the computation complexity but also guarantees integer solutions if the capacities of the edge are integers. However, the main drawback is that the net ordering problem dominates in its flow generating algorithm.

R.C. Carden [Card91] developed a multiterminal, multicommodity flow algorithm for their global router. This approach is based on Shahrokhi and Matula's epsilon bound algorithm [Shah88] which yields the fractional flow solution in a relatively short execution period. First a discrete net connection with an error bound is produced from the optimal fractional solution. Next, an iterative procedure is used to improve the final results. However, due to the high computation complexity, this algorithm may not be solved in polynomial time. Furthermore, this approach can not guarantee an optimal solution.

Yet another linear programming (LP) related approach was presented in [Cong98]. This global router continuously decomposes the whole global routing problem into smaller subproblems and allows at most four bends for each net. For each level, a solution can be found by a two-stage approach of small sized LP followed by min-cost flow networks. First, the routing area is partitioned into four square sub-area and a Linear Programming algorithm is utilized to compute the density contribution on the cutting edges. Next, the problem is formulated as a network flow and solved by a min-cost flow technique [Sarr94]. The running time is reduced by using the two-stage algorithm. Since only four-bend nets are allowed, the problem size of the LP at each level is also reduced.

3.3 Summary

In this chapter, several approaches for global routing were reviewed. Two classes of techniques, namely sequential approaches and concurrent approaches were identified and discussed. Sequential approaches are able to solve global routing problems efficiently. However, they suffer from the net-ordering problem. This problem is avoided via concurrent approaches by routing nets simultaneously. From this brief review, two sequential heuristic approaches in addition to an Integer Linear Programming (ILP) based approach are to be implemented in the next chapter.

Chapter 4

Heuristic/ILP-based Routing Algorithms

Global routing is an NP-hard problem that cannot be solved exactly in polynomial time [Blan85]. Therefore, heuristic methods are used to find a good solution in reasonable time. This chapter presents a Sequential Heuristic Global Router (SHGR) which can effectively solve the global routing problem. Based on SHGR, a novel Hierarchical Heuristic Global Router (HHGR) is implemented by using the multi-level clustering technique which can provide routing estimations in reasonable time.

Heuristic approaches usually suffer from the net-ordering problem. Although good performance can be achieved by SHGR and HHGR, it is hard to find a good net order that guarantees an optimal solution [Cong92]. In addition, only 2-bend routes are allowed which underestimate the actual wire length.

Therefore, an Integer Linear Programming based global router (ILP-GR) is also proposed in this chapter. The ILP-GR considers all nets simultaneously to avoid the net-ordering problem. Furthermore, it connects nets by constructing Minimum Rectilinear Steiner Trees (MRST), which produce shorter wire length.

All the algorithms are tested on Sun-blade 2000 workstation under Solars operating system. Statistics of several MCNC benchmarks were introduced in section 2.5. These benchmarks are selected since they are widely used in verifying VLSI physical design algorithms to solve partitioning, placement and routing problems.

4.1 A Sequential Heuristic Global Router (SHGR)

4.1.1 Cost Array Routing Model

In order to solve the standard cell global routing problem, the standard cell layout is represented as a *Cost Array* routing model [JRos88]. As illustrated in Figure 4.1, each possible routing position in a channel is an element in the *Cost Array*. The vertical dimension is set to the number of rows plus 1 and its horizontal dimension is set to the width of the routing grid. Let P_{ij} be the position at channel i and routing cell j. There are two constraints for each element in the *Cost Array*:

- H_{ij}: The number of horizontal wires passing through P_{ij}.
- V_{ij}: The number of vertical wires traversing P_{ij}.

The total cost of a path P can then be defined as:

$$Cost(P) = \sum_{P}(H_{ij} + V_{ij}) \tag{4.1}$$

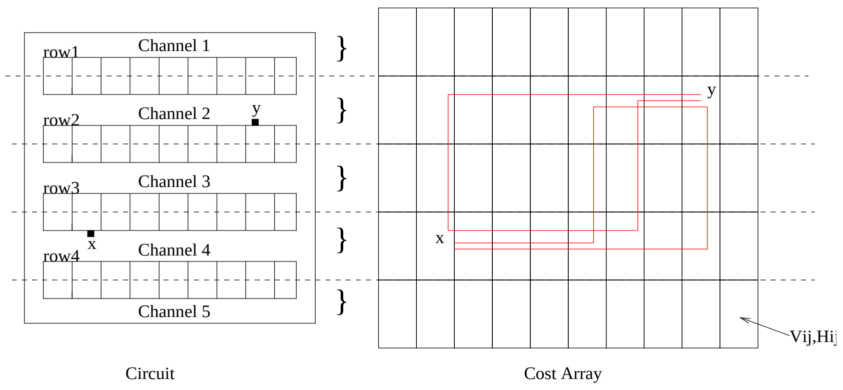


Figure 4.1: Cost Array model

4.1.2 SHGR Implementation

The main objective of SHGR is to find the minimum cost path for each net. Its pseudo-code is illustrated in Figure 4.2. For each two-terminal net, the minimum cost path is selected by exploring a set of all possible 2-bend routes [JRos88]. For the remaining multi-terminal net, the wire is first decomposed into several two-point segments using a Minimum Spanning Tree (MST) algorithm. Next the minimum cost path is determined for every two-point segment. The complexity of SHGR is $O(WS)$, where W is number of wires and S is the number of the two-point segments. The detailed SHGR algorithm is discussed as follows:

1. For a multi-terminal net, a Minimum Spanning tree (MST) is constructed to

```
     Sequential Heuristic Algorithm
1. Start
2. Read Circuit information and build cost array;
3. While (wire queue not empty)
4.    If (2-terminal wire)
        Explore all possible 2-bend routes;
        Find the minimum cost path;
5.    Else
        Decompose wire into 2-point segments;
        Forall 2-point segments
           Explore all possible 2-bend routes;
           Find the minimum cost path;
        End Forall
        Combine all 2-point paths back together;
6.    End If
7. End While
8. End
```

Figure 4.2: Sequential Heuristic Algorithm

connect all pins. Each edge in this MST is considered as a two-point segment. As illustrated in Figure 4.3, a 5-terminal wire is decomposed into a set of 4 two-point segments: $\{A, B\}$, $\{A, C\}$, $\{A, D\}$ and $\{D, E\}$

2. The two-point segments are further decomposed into permutations. In the standard cell design style, each pin has its electrically equivalent pin which is usually on the opposite side of the cell (as demonstrated previously in Figure 2.16). Connecting either pin will produce the same electrical output (such a pin pair is called a cluster). A permutation is a route between the two clusters of a two-point segment. As illustrated in Figure 4.4, cluster A has a pair of equivalent pins $A1$ and $A2$, cluster B has a pair of equivalent pins $B1$ and $B2$. Between the two clusters, there are four possible permutations $p1$, $p2$, $p3$ and $p4$. These four permutations have the same electrical output, but $p2$ has

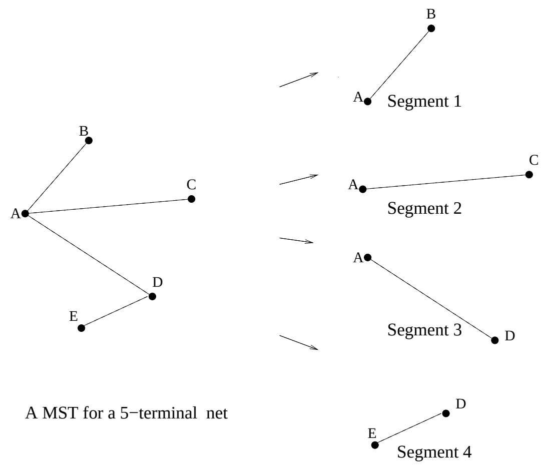


Figure 4.3: A 5-point net is decomposed into four 2-point segments

the minimum cost.

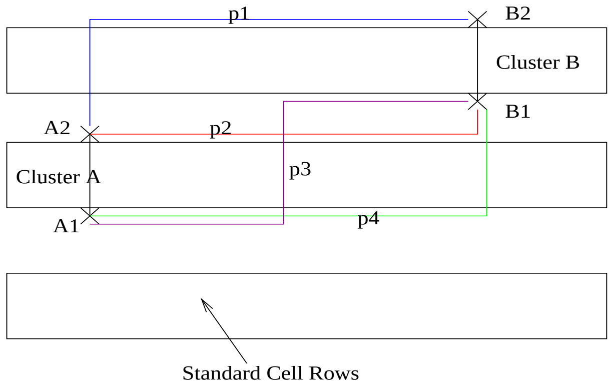


Figure 4.4: Four Possible Permutations Between Cluster A and Cluster B

3. A low-cost path in the cost array, in which each element represents a possible routing position, is found for each permutation by evaluating a subset of the two-bend routes [AN87] between each pin pair. The process of constructing two-bend routes is illustrated in Figure 4.5.

   Given a two-point segment (provided the coordinates of two points are left-

```
2-bend Algorithm
1. Start
2. For (row=0; row < R; row++)
     wire1_start=(0,0); wire1_end=(0,row);
     wire2_start=(0,row); wire2_end=(C,row);
     wire3_start=(C,row); wire3_end=(C,R);
3. End For loop 1
4. For (col=0; col < C; col++)
     wire1_start=(0,0); wire1_end=(col,0);
     wire2_start=(col,0); wire2_end=(col,R);
     wire3_start=(col,R); wire3_end=(C,R);
5. End For loop 2
6. End
```

Figure 4.5: 2-bend Algorithm to find routes between point (0,0) and (C,R)

most pin $P(0,0)$ and right-most pin $P(C,R)$), each two-bend connection normally consists of 3 wire segments combined together: $wire1$, $wire2$ and $wire3$. These wire segments can be determined by $loop_1$ and $loop_2$. $Loop_1$ generates the principally horizontal routes (bend only at the left and right extremes). Principally horizontal routes are first evaluated since they make the best use of the channels. When $wire1$ has the same row number as point $P(0,0)$, its start position and end position are (0,0). $Wire1$ is actually a point, therefore only $wire2$ and $wire3$ exist in this two-bend wire. $Loop_2$ generates the principally vertical routes (bend only at the upper and lower extremes). When $wire1$ has the same column number as point $P(0,0)$, it is a point and there are only two wire segments in this two-bend wire. The complexity of this algorithm is $O\left(Max(C,R)\right)$, where C is the number of columns and R is the number of rows.

Figure 4.6 gives an example of the result of constructing all possible 2-bend routes between two pins in a 2×3 grid graph. Figure 4.6 (a) shows the principally horizontal routes, and Figure 4.6 (b) shows the principally vertical routes.

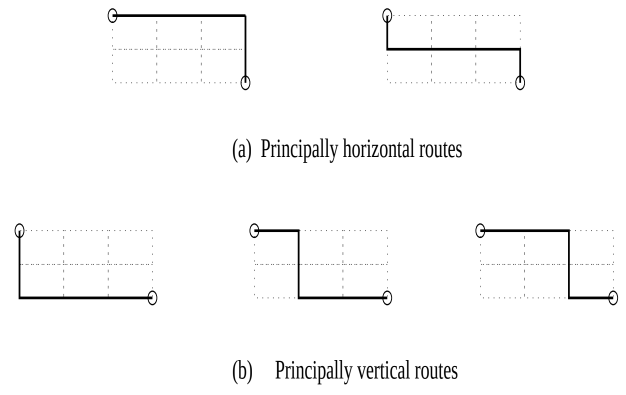

(a) Principally horizontal routes
(b) Principally vertical routes

Figure 4.6: (a)Routes Generated by Loop 1 (b)Routes Generated by Loop 2

4. All 2-bend routes produced for each segment are traced back. They are combined together to form the final connection of the MST tree which connects all pins in the netlist.

5. The wire is put into the cost array such that array elements are updated and later wires can take the cost constrains into account.

4.1.3 Experimental Results

Table 4.1 lists the wire lengths and execution time yielded by SHGR. It is tested on MCNC benchmarks from the small circuits such as *fract* and *struct* to large circuits such as *avq.small* and *avq.large*.

Circuit	Nnets	WireLength (μ)	time (s)
fract	147	21822	0.1
struct	1920	448438	2.2
prim1	1133	891798	4.5
ind1	2478	1650600	130
prim2	3817	4201487	150
bio	5742	6327872	326
ind2	13419	50216723	1411
ind3	21938	93136107	1734
avq.small	22124	7739168	12679
avq.large	25384	8477116	12639

Table 4.1: Wirelength and time of SHGR

Circuit	Placement Tool	SHGR	Improvement (%)
fract	33187	21822	34.2 %
struct	494832	448438	9.3 %
prim1	930533	891798	4.2 %
ind1	2130137	1650600	22.5 %
prim2	6192874	4201487	32.1 %
bio	6555340	6327872	3.5 %
ind2	54470116	50216723	7.8 %
ind3	100886561	93136107	7.7 %
avq.small	9646996	8023029	15.2 %
avq.large	11023017	8477116	23.1 %
average	19236359	17311113	16.0 %

Table 4.2: Wire length comparison between the placement CAD tool and the SHGR

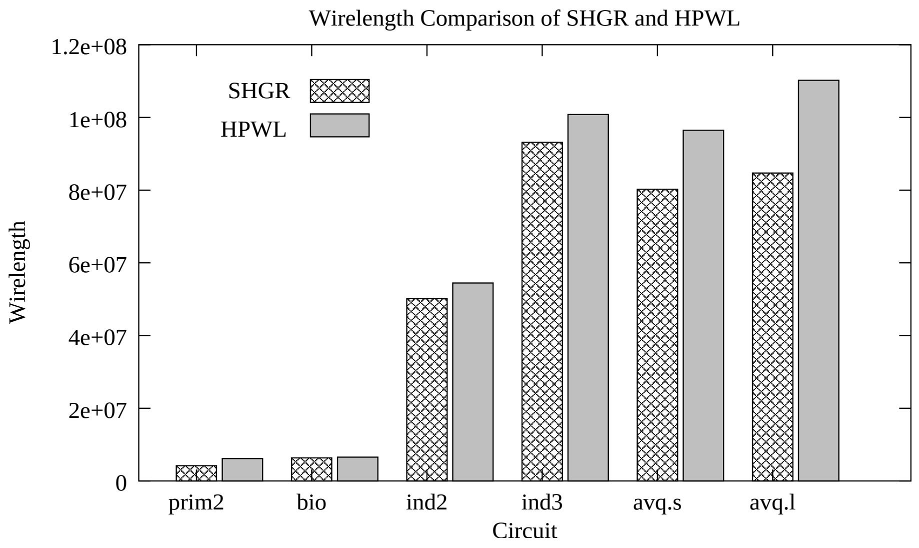


Figure 4.7: Wirelength comparison of SHGR and HPWL

Table 4.2 and Figure 4.7 show that SHGR achieves an improvement of 16% over wirelength yielded by placement results based on HPWL. As discussed in chapter 2, HPWL method underestimates the actual wirelength. However, SHGR still produces shorter wirelength than HPWL based global router, since it considers the position of equivalent pins. The pin can be connected through its corresponding equivalent pin on the other side of the cell. Therefore the distance is possibly shorter than the path routed by HPWL, which only connects the pin through the actually pin location.

From the experimental results, it can be concluded that SHGR produces better solutions than the placement based on HPWL method. The shorter wirelength yielded and reasonable CPU running time make it an efficient and useful global router for VLSI circuits from small sized to large sized problem.

4.2 A Hierarchical Heuristic Global Router (HHGR)

As the complexity of a VLSI circuit increases, a hierarchical improvement approach becomes essential to shorten the design process. As stated in section 2.1, circuit clustering plays a fundamental role in hierarchical designs. Identifying highly connected components in the netlist can significantly reduce the complexity of the circuit and improve the design time and quality.

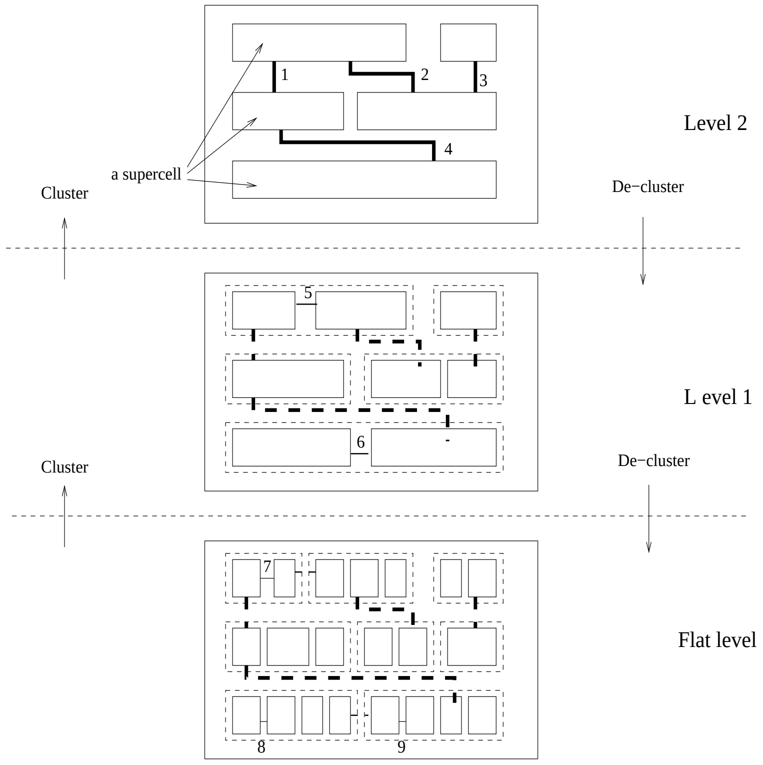


Figure 4.8: A 3 level cluster seed placement

A Hierarchical Heuristic Global Router (HHGR) is implemented in this thesis which is based on the Sequential Heuristic Global Routing technique. The main objective is to find the minimum cost path by enumerating a set of possible 2-bend routes. However, HHGR routes a subset of wires at different hierarchical levels. For example, as illustrated in Figure 4.8, the number of hierarchical levels used is 3. At $level_3$, there are 4 wires: $wire_1$, $wire_2$, $wire_3$ and $wire_4$. HHGR only needs to find the minimum cost paths for these 4 wires. Next at $level_2$, only $wire_5$ and $wire_6$ need to be routed. Finally at the $flat\ level$, the same routing process continues, and the remaining wires ($wire_7$, $wire_8$ and $wire_9$) are routed by HHGR.

4.2.1 HHGR Implementation

```
Hierarchical Heuristic Algorithm
1. Start
2. Circuit clustering;
3. For (level=N; level ≥ 0; level−−)
4.     While (wire queue not empty)
5.       If (all pins in one super cell)
           Wire is hold up to next level;
6.       Else if (only some pins in one super cell)
           Decompose wire into several segments;
           Use SHGR to route the segments within the current level;
           Other wire segments are hold up to the next level;
7.       Else
           Use SHGR to route wire;
8.       End If
9.     End While
10. End For loop
11. End
```

Figure 4.9: Hierarchical heuristic algorithm

The pseudo-code of HHGR is presented in Figure 4.9. Following hierarchical placement, N hierarchical levels are obtained using the multi-level clustering technique. In a top-down direction, at each level, HHGR retrieves wires from a *wire queue*. If the locations of all pins are in one super cell, this wire will be passed

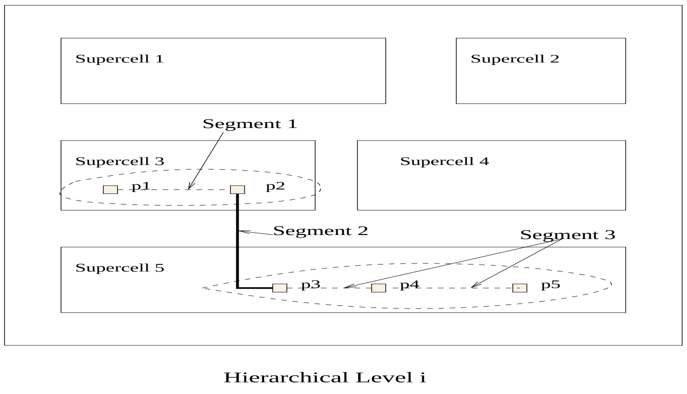


Figure 4.10: A net is cut into segments

to the next lower level. If only some pins are in one super cell, the wire will be cut into several segments. Some segments are held until the next level while others are routed by SHGR at the current level.

Figure 4.10 shows an example of cutting a 5-pin wire into 3 segments at $level_i$. There are 5 super cells at the current hierarchical level i. Two pins $p1$ and $p2$ are in $super\text{-}cell_3$, and pins $p3$, $p4$ and $p5$ are in $super\text{-}cell_5$. Therefore, the wire is cut into three segments: S_1 : {p1, p2}, S_2 : {p2, p3} and S_3 : {p3, p4, p5}. Pins in S_1 and S_3 are located in their own super cells. Thus these two segments are passed to the next $level_{i-1}$, while SHGR is used to find the minimum cost path for S_2.

At $level_{i-1}$, the same process continues until the $flat\ level$ representation is reached. At the $flat\ level$ all wires are routed and the minimum cost 2-bend routes are found for each wire until the wire queue is empty.

Circuit	Heuristic	Hierarchical	Improvement(%)
prim1	4.5	0.3	94 %
ind1	130	2	83 %
prim2	150	6	96 %
bio	326	30	91 %
ind2	1411	62	95 %
ind3	1734	108	91 %
avq.small	12679	2565	80 %
avq.large	12639	2243	82 %
average	3634.2	423.7	89 %

Table 4.3: CPU Time comparison between SHGR and HHGR

4.2.2 Experimental Results

Table 4.3 and Figure 4.11 compares HHGR with SHGR (3 hierarchical levels) in terms of CPU processing time.

Circuit	Heuristic	Hierarchical	Improvement (%)
prim1	891798	1013446	-13.6 %
ind1	1650600	1929884	-16.9 %
prim2	4201487	4418238	-5.2 %
bio	6327872	6593101	-4.2 %
ind2	50216723	50032892	0.3 %
ind3	93136107	98364933	-5.6 %
avq.s	7739168	9519822	-23.0 %
avq.l	8477116	9836109	-16.0 %
average	20626434	22713553	-9.2 %

Table 4.4: Wirelength comparison between SHGR and HHGR

For some small MCNC benchmarks such as $fract$ and *struct*, since the CPU times used by both routers are very small, they are not listed in this section. As seen in Table 4.3, obviously HHGR is faster than SHGR. The CPU time is reduced

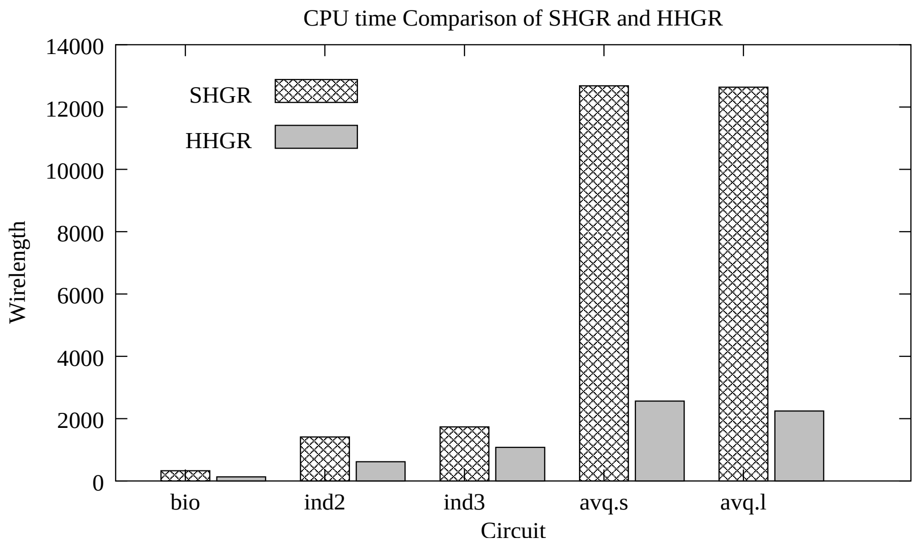


Figure 4.11: CPU time comparison of SHGR and HHGR

by 89% on average. Since the construction of MSTs in SHGR has the largest computation time in the heuristic, by cutting a large net into several small ones and routing them at different levels can dramatically shorten the execution time.

However, wirelengths produced by HHGR are on average 9.2% longer than SHGR, as seen in Table 4.4 and Figure 4.12. Since constructing MSTs for different portions of a net can not guarantee the same optimal solution as that obtained by constructing one MST for the whole net.

Although HHGR degrades the solution quality by producing longer wirelength, it is still a useful global router due to the short CPU time. In some cases, circuits are not able to be routed because of bad placement output. Therefore it is important to predict the routability of a circuit in a very short time before the routing phases starts. HHGR is suitable as an estimation models since it give a reasonable routing solution in a short time. Thus it can be used as a pre-processing tool during

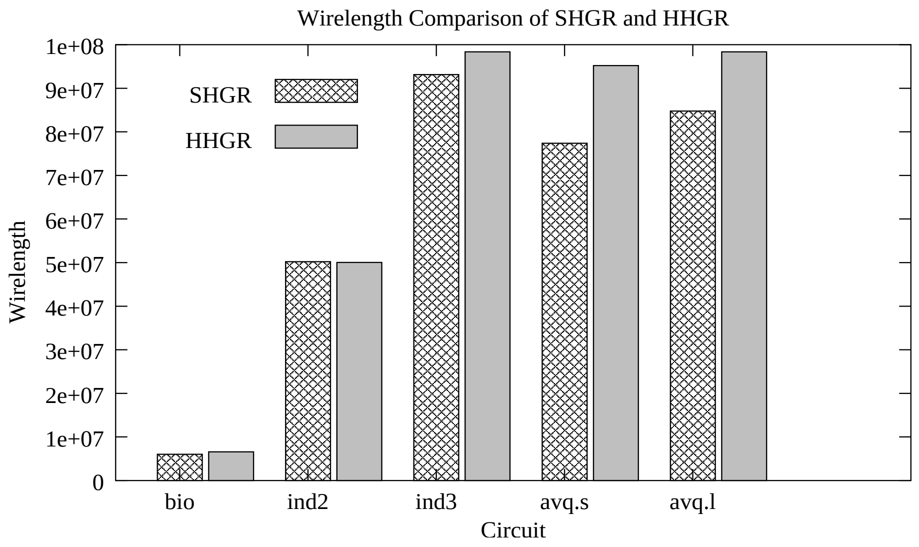


Figure 4.12: Wirelength comparison of SHGR and HHGR

hierarchical placement to provide routing estimation information.

4.3 An ILP Based Global Router (ILP-GR)

An ILP Based Global Router (ILP-GR) is proposed here to avoid the net-ordering problem that other sequential routers usually suffer from. Furthermore, ILP-GR produces shorter wire length by constructing Minimum Rectilinear Steiner Trees (MRST).

An ILP-GR is implemented as follows: First, a set of MRSTs are generated and a binary variable is assigned to each tree. Next, an Integer Linear Programming (ILP) model is formulated with all the necessary constraints. To further reduce computational complexity, integer constraints are relaxed and the problem is solved as a Linear Programming (LP) model. Finally, the relaxed LP problem is solved by using an LP solver from ILOG (CPLEX) [SA01].

4.3.1 Minimal Rectilinear Steiner Tree (MRST)

The Minimum Steiner Tree problem can be defined as follows: Given a set P of n points, find a set S of *Steiner points* such that the Minimum Spanning Tree (MST) over $P \cup S$ has minimum cost.

In most cases, since rectilinear geometry is used for circuit layout, the Minimum Rectilinear Steiner Tree (MRST) is the most widely accepted connection pattern. Usually it can be derived from a Minimum Spanning Tree (MST). Hwang [Hwan76] has shown that the ratio of the cost of an MST to that of an optimal MRST is no greater than $\frac{3}{2}$. Figure 4.13 shows an MST and an MRST for the same four-pin net.

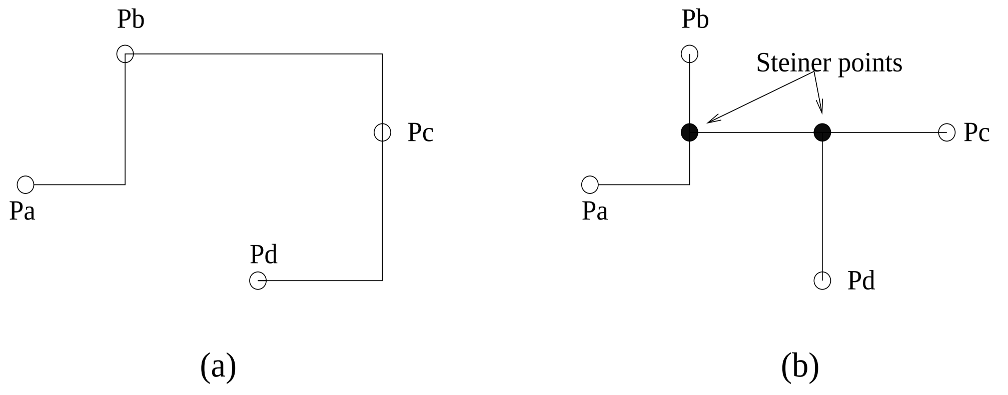


Figure 4.13: (a)A Minimum Spanning Tree (MST) for a fixed net. (b)A Minimum Rectilinear Steiner Tree (MRST) for a fixed net.

4.3.1.1 Construction of Minimal Rectilinear Steiner Trees

For each given net, provided the maximum number of Steiner points is Max_S, a recursive function *Find_SteinerTree()* is developed to generate the MRST. The possible Steiner points are added to the point set iteratively until there are no more available Steiner points left. The algorithm is illustrated in Figure 4.14.

```
                Steiner Tree Generation
1. Start
1. Read subgrid information;
3. while (S $\leq$ Max_S)
       Find_SteinerTree(Row, Col, S, S+P);
       S=S+1;
4. End While
5. End
```

Figure 4.14: Algorithm to find a set of Steiner trees

The *Find_SteinerTree()* procedure visits all the candidate Steiner points s in set S and all the pins in set P. Its main objective is to find a spanning tree in the point set $P \cup S$. As illustrated in Figure 4.15, it uses the *getNextSteinPoint()* procedure to compute the next available Steiner point s in set S. If Steiner points are available, the recursive loop continues. Otherwise, the function *Find_MST()* takes the point set $P \cup S$ as input and returns its corresponded Minimum Spanning tree. The complexity of the *Find_SteinerTree()* is $O\left((P+S)^2\right)$, and the complexity of finding a set of Steiner Trees is $O\left(Max\_S(P+S)^2\right)$.

```
                 Find_SteinerTree()
1. Start
2. S=S-1;
3. While (get_nextSteinPoint(currRow,currCol))
4.    If (S > 0)
        Find_SteinerTree(currRow, currCol, S, S+P);
5.    Else
        Found_A_Tree=do_MinTree(S+P);
        If (Found_A_Tree)
           add_to_TreeSet();
           Find the minimum cost path;
        End If
6.    End If
   End While
7. End
```

Figure 4.15: Function Find_SteinerTree()

4.3.1.2 Size of the Steiner Tree Set

The number of Steiner trees generated for each net depends on terminals in the net. It is impossible to include all the possible Steiner trees for point set $P \cup S$. Generating all possible Steiner trees for a net results in a large number of variables and constraints, which enlarges the optimization problem size. Moreover, the problem of finding a Steiner tree is NP-hard and consequently computation time increases exponentially as the size of the point set $P \cup S$ increases [Gare79]. Practically, only a subset of the whole Steiner tree set needs to be explored.

Finding an appropriate size of the Steiner tree subset is critical and therefore two approaches have been investigated. The first ($Steiner\_ILP1$) attempts to construct trees proportional to the terminals in the net. The second approach ($Steiner\_ILP2$) is more conservative where only 10 trees are generated for all types of nets.

Table 4.5 highlights the number of nets and their composition in terms of number of terminals for MCNC benchmark circuits. For *fract* circuit the total number of trees generated by the approach $Steiner\_ILP1$ is: $(10 \times 70) + (20 \times 44) + (30 \times 14) + (40 \times 13) + (50 \times 6) = 2820$. On the other hand the number of trees generated by the approach $Steiner\_ILP2$ is limited to 1470. Table 4.6 shows the corresponding number of trees generated for all circuits. Obviously, the tree set generated by $Steiner\_ILP1$ is much larger than $Steiner\_ILP2$ but has higher potential of achieving shorter wirelength. However, the main disadvantage is the high computation time. It is clear from Table 4.6 that although the small tree set degrades the solution quality, the running time to generate Steiner Trees is shortened by almost 50%. Therefore, considering the time and quality factor, the approach $Steiner\_ILP2$ is selected to generate the Steiner Trees in this thesis. The number of trees is fixed as 10 since experimental results show tree subset in this size produces good solution qualities in a reasonable time.

Circuit	Nnets	# 2-terminal	# 3-terminal	# 4-terminal	# 5-terminal	# others
fract	147	70	44	14	13	6
struct	1920	737	1151	0	0	32
prim1	876	463	235	70	27	81
ind1	2478	1601	401	150	78	248
prim2	3136	1942	365	203	192	434
bio	5742	3998	870	427	184	263
ind2	13419	9585	1851	294	447	1242
ind3	21938	12438	4973	1923	1518	1086
avq.small	22124	13654	6136	1478	712	144
avq.large	25384	16914	6136	1478	712	144

Table 4.5: Number of pins in the nets for MCNC benchmark circuits

Circuit	Approach Steiner_ILP1			Approach Steiner_ILP2		
	Ntrees	Wirelength(μ)	Time(s)	Ntrees	Wirelength(μ)	Time(s)
fract	2820	21475	0.7	1470	21840	0.3
struct	30390	398771	32	19200	402382	5.7
prim1	15560	807580	60	8760	830365	9.1
ind1	44050	1474582	398	24780	1509809	169
prim2	62190	4087895	779	31360	4151385	197
bio	90700	5214987	2498	57420	5266555	488
ind2	221670	42755375	6287	134190	45032892	2175
ind3	396550	90467584	7259	1219380	92599062	3042
avq.small	339820	6938163	28573	221240	7085894	15271
avq.large	371880	7627771	29550	253840	7762927	15423

Table 4.6: Comparison of $Steiner\_ILP1/Steiner\_ILP2$ to generate Steiner Trees

4.3.2 ILP Formulation

The Integer Linear Programming (ILP) formulation of the global routing problem is adapted and modified from the work of [Vann91]. Two ILP models are formulated in this thesis which are the "net connection model" and the "channel density model" respectively. A CPLEX interface to the current placement tool (SC3) is implemented to automate the process.

4.3.2.1 Net Connection Model

Wire length is an important factor to evaluate the performance of a global router. The objective of the net connection ILP model is to maximize the net connection benefit such that the total wire length is shortened. Let N be the set of all nets, for each net $i \in N$, there are several possible connections, where each is a tree in a set of rectilinear Steiner trees $T_i^1, \ldots, T_i^k$. A variable y_i is associated with each tree which connects a net. It is set to "1" if that particular tree is used, otherwise it is set to "0". However, only one tree is finally selected to connect this net. The selection of one and only one routing tree for this net is imposed by the following constraint:

$$\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N|. \tag{4.2}$$

A (0,1) matrix $[a_{ij}]$ is used to record all possible trees to connect nets. Given a grid graph with m arcs, the number of rows is set to m. The ith row represents the corresponding ith arc in the grid graph and each column corresponding to different ways of connecting a net. The a_{ij} element in matrix A can be expressed as:

$$a_{ij} = \begin{cases} 1 & \text{if tree } j \text{ is constructed by arc } i \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The main objective of ILP-GR is to find the possible connection patterns to connect all nets without violating the resource capacity constraints. Let n be the number of all generated Steiner Trees. For each net, assume the capacity on the edge e_i is c_i, the net connection ILP formulation can be expressed as:

$$Maximize \qquad \sum_{j=1}^{n} b_j y_j. \tag{4.4}$$

subject to

$$\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N| \qquad (ROUTING)$$
$$\sum a_{ij} y_j \leq c_i, \qquad 1 = 1, 2, \ldots m$$
$$y_j \in \{0, 1\}$$

Where b_j is the benefit of connecting a net by tree y_j. Maximizing the net connection benefits eventually leads to minimum wirelength which in place yields higher performance chips. The value for b_j can be determined by wirelength. Let $MaxWirelength$ be the maximum wirelength for all trees in all nets and b_j defined as:

$$b_j = (MaxWirelength + 1) - (wirelength\ of\ tree\ j) \tag{4.5}$$

Figure 4.16 shows a simple global routing formulation example. The circuit has 3 modules and 2 nets. Net_1 contains modules 1, 2 and 3, and Net_2 contains modules 1 and 2. There are 7 edges (e_1, e_2 e_3, e_4, e_5, e_6 and e_7) with capacities (c_1, c_2, c_3, c_4, c_5, c_6 and c_7).

Net_1 can be routed by four trees:

$Tree_1$: (y_1) is connected by edges e_1, e_2, e_3, e_6, wirelength is 4;

$Tree_2$: (y_2) is connected by edges e_2, e_5, e_7, wirelength is 3;

$Tree_3$: (y_3) is connected by edges e_2, e_4, e_6, wirelength is 3;

$Tree_4$: (y_4) is connected by edges e_1, e_3, e_4, e_6, wirelength is 4.

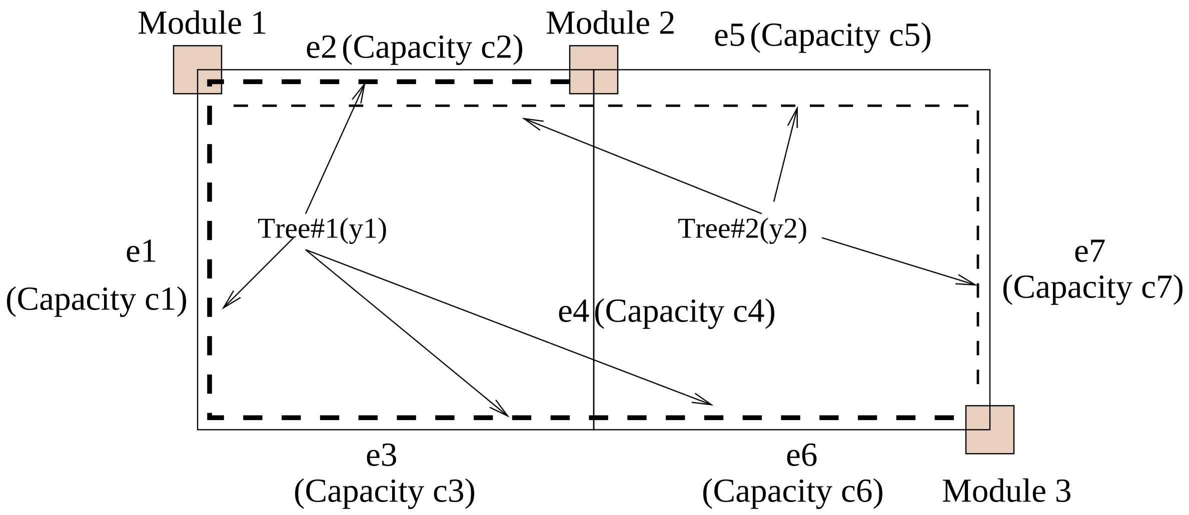


Figure 4.16: Example of the net connection model

Net_2 can be routed by two trees:

$Tree_1$: (y_5) is connected by edges e_2, wirelength is 1;

$Tree_2$: (y_6) is connected by edges e_1, e_3, e_4, wirelength is 3;

The values for b_j can be calculated by using equation 4.5. Thus the net connection model can be formulated as:

$$Max \qquad y_1 + 2y_2 + 2y_3 + y_4 + 4y_5 + 2y_6$$

subject to:

$$\begin{aligned} y_1 + y_2 + y_3 + y_4 &\leq 1 \\ y_5 + y_6 &\leq 1 \\ y_1 + y_4 + y_6 &\leq c_1 \\ y_1 + y_2 + y_3 + y_5 &\leq c_2 \end{aligned}$$

$$\begin{aligned}
y_1 + y_4 + y_6 &\leq c_3 \\
y_3 + y_4 + y_6 &\leq c_4 \\
y_2 &\leq c_5 \\
y_1 + y_3 + y_4 &\leq c_6 \\
y_2 &\leq c_7 \\
y_j &\in \{0, 1\}
\end{aligned}$$

4.3.2.2 Channel Density Model

For a standard cell circuit, the width of the chip is already determined by the maximum of the row widths, therefore the area of the chip is minimized by reducing the area between rows [Sher99]. As previously stated in section 2.3, such routing areas are measured by the channel density. The objective of the channel density ILP model is to minimize the total channel density such that the produced chip area is optimal.

Let C be the number of channels and the channel densities be z_1, ..., z_C respectively which are integers in $(0, D)$. Thus the objective function of ILP formulation can be expressed as:

$$Minimize \qquad \sum_{k=1}^{C} z_k \tag{4.6}$$

For each net there are several possible connection trees, the selection of one and only one routing tree for it is same as the equation 4.2 used in net connection model:

$$\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N|. \tag{4.7}$$

A (0,1) matrix $[a_{ij}]$ is used to record all possible trees vertically passing through a position in the *cost array*. Given m arcs, the number of rows of matrix $[a_{ij}]$ is set to m and the value of each arc is calculated in the *cost array*. The ith row represents the corresponding ith arc in the grid graph and each column corresponding to different ways of connecting a net. The a_{ij} element in matrix A can be expressed as:

$$a_{ij} = \begin{cases} 1 & \text{if tree } j \text{ is constructed by arc } i \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

Similar to the matrix $[a_{ij}]$, a (0,1) matrix $[b_{ij}]$ is used to record all possible trees which may use one of the tracks. Given the number of track arcs is h, The b_{ij} element in matrix B can be expressed as:

$$b_{ij} = \begin{cases} 1 & \text{if tree } j \text{ is constructed by track arc } i \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Therefore, the formulation to minimize the total channel density can be expressed as:

$$Minimize \qquad \sum_{k=1}^{C} z_k \tag{4.10}$$

subject to

$$\begin{aligned}
&\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N| \qquad (ROUTING)\\
&\sum a_{ij} y_j \le c_i, \qquad i = 1, 2, \ldots m\\
&\sum b_{ij} y_j \le z_k, \qquad i = 1, 2, \ldots h\\
&(or \ \sum b_{ij} y_j - z_k \le 0, \qquad i = 1, 2, \ldots h)\\
&y_j \in \{0, 1\}\\
&0 \le z_k \le D, \qquad z_k \ is \ an \ integer
\end{aligned}$$

To illustrate the channel density model, as seen in Figure 4.17, modules 1 and 2 are placed in the first row, and module 3 is placed in the second row. Edges e_2 and e_5 form the $channel_1$ and edges e_3 and e_6 form the $channel_2$. The two nets Net_1, Net_2 and six trees y_1, y_2, y_3, y_4, y_5 and y_6 are similar to those in Figure 4.16.

Assume the number of tracks in each channel (z_1 and z_2) can not exceed 2, the channel density model to minimize the number of tracks can be expressed as:

$$Min \qquad z_1 + z_2$$

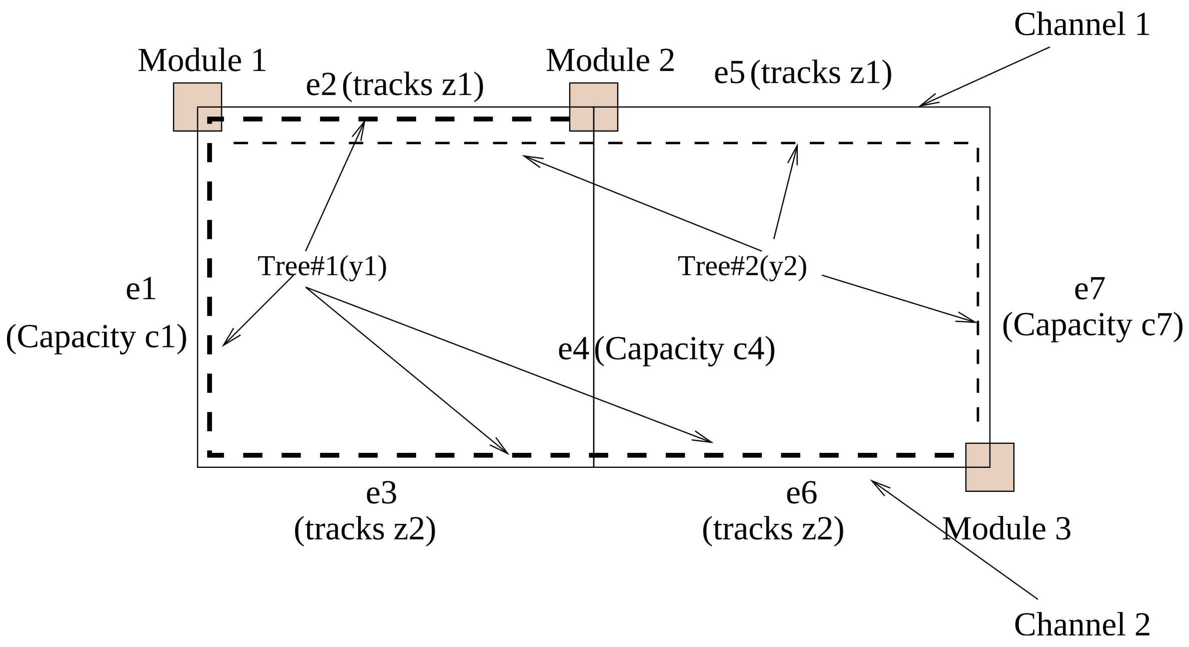


Figure 4.17: Example of the channel density model

subject to:

$$\begin{aligned}
y_1 + y_2 + y_3 + y_4 &\leq 1 \\
y_5 + y_6 &\leq 1 \\
y_1 + y_4 + y_6 &\leq c_1 \\
y_3 + y_4 + y_6 &\leq c_4 \\
y_2 &\leq c_7 \\
y_1 + y_2 + y_3 + y_5 &\leq z_1 \\
y_1 + y_4 + y_6 &\leq z_2 \\
y_2 &\leq z_1 \\
y_1 + y_3 + y_4 &\leq z_2
\end{aligned}$$

$$\begin{aligned} y_j &\in \{0,1\} \\ 0 \leq z_1 &\leq 2, \qquad z_1 \textit{ is an integer} \\ 0 \leq z_2 &\leq 2, \qquad z_2 \textit{ is an integer} \end{aligned}$$

4.3.2.3 Experimental Results

The objective of $Model_1$ (net connection model) is to produce the shortest wirelength by maximizing the net connection benefit. However in order to get the shortest wirelength, the yielded chip area is expected to be suboptimal. On the other hand, $Model_2$ (channel density model) minimizes the total channel density and chip area, while producing wirelengths that are suboptimal.

As seen in Table 4.7, the wirelength produced by $Model_2$ (channel density model) is longer than $Model_1$ (net connection model). It shows that the net connection model tends to choose routes which yield shorter wirelengths instead of smaller channel densities. Table 4.8 compares the total channel density produced by $Model_1$ and $Model_2$. Obviously, $Model_2$ (channel density model) produces smaller channel density.

Circuit	Model 1	Model 2	Difference(%)
prim1	830365	841537	1.34 %
ind1	1509809	1520373	0.7 %
prim2	4151835	4188721	0.89 %
bio	5266555	5367362	1.91 %
ind2	45032892	45524948	1.09 %
ind3	92599062	92643742	0.05 %
avq.small	7085894	7234845	2.10 %
avq.large	7762927	7873837	1.43 %

Table 4.7: Comparison of wire length between Model 1 and Model 2

Circuit	Model 1	Model 2	Difference(%)
prim1	178	174	2.30 %
ind1	454	450	0.89 %
prim2	1098	1082	1.46 %
bio	592	582	1.72 %
ind2	1442	1426	1.11 %
ind3	2579	2553	1.02 %
avq.small	1372	1358	1.03 %
avq.large	1704	1687	1.00 %

Table 4.8: Comparison of the number of tracks between Model 1 and Model 2

The net connection ILP model and channel density ILP model emphasis different objectives for the global routing problem. The selection of ILP models depends on the performance requirements given in the specification. If the wirelength is the main factor to be evaluated, the net connection ILP model is used, otherwise the channel density ILP model is selected. In the future, the two models can be combined together as a single multi-objective ILP model. The combined ILP formulation targets at finding optimal solutions that seek to shorten the wirelength and minimize the channel density at the same time.

4.3.3 Linear Relaxation

The CPU time required to solve the ILP problem increases exponentially as the problem increases in size since it is NP-hard. In order to reduce the computation complexity, the formulated ILP problem can be relaxed as a Linear Programming (LP) problem. The LP relaxation is performed by replacing integer constraints on the variable with linear boundary constraints.

4.3.3.1 Relaxed Net Connection Model

Based on definition 4.4, by removing integer constraints on variable y_j, the relaxed net connection formulation is defined as:

$$Maximize \qquad \sum_{j=1}^{n} b_j y_j. \tag{4.11}$$

subject to:

$$\begin{aligned}
&\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N| \qquad (ROUTING) \\
&\sum a_{ij} y_j \leq c_i, \qquad 1 = 1, 2, \ldots m \\
&0 \leq y_j \leq 1
\end{aligned}$$

For the global routing problem, each y_j is a non-negative variable, therefore the constraint equation $0 \leq y \leq 1$ is redundant since it is enforced by the constraint $\sum y_j = 1$. To further reduce the number of constraints, the LP problem can be expressed as:

$$Maximize \qquad \sum_{j=1}^{n} b_j y_j. \tag{4.12}$$

subject to:

$$\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N| \qquad (ROUTING)$$

$$\sum a_{ij}y_j \leq c_i, \qquad 1 = 1, 2, \ldots m$$
$$0 \leq y_j$$

4.3.3.2 Relaxed Channel Density Model

Similarly, by replacing integer constraints on variables y_j and z_k with linear boundary constraints, based on definition 4.10, the relaxed channel density formulation is defined as:

$$Minimize \qquad \sum_{k=1}^{C} z_k \tag{4.13}$$

subject to

$$\sum_{y_j \in T_k} y_j = 1, \qquad k = 1, 2, \ldots |N| \qquad (ROUTING)$$
$$\sum a_{ij}y_j \leq c_i, \qquad i = 1, 2, \ldots m$$
$$\sum b_{ij}y_j - z_k \leq 0, \qquad i = 1, 2, \ldots h$$
$$0 \leq y_j$$
$$0 \leq z_k \leq D$$

4.3.3.3 Experimental Results

Solutions of the relaxed LP models may be non-integral. The fractional solution y_j for each net represents how much of tree y_j is used to route the net. These fractional values have no physical meaning, and therefore the y_j for each net has to be rounded to an integer value. If the difference between ILP and its relaxed LP solution is small, the rounded LP solution can be used to replace the ILP solution.

Circuit	ILP formulation		LP Relaxation	
value	Wirelength(μ)	Time(s)	Wirelength(μ)	Time(s)
prim1	830365	0.5	830365	0.4
ind1	1509809	0.6	1509809	0.4
bio	4151835	1.2	4151835	0.8
prim2	5266555	8.1	5266555	6.5
ind2	45032892	28.1	45032892	21.6
ind3	92599062	78.2	92599062	56.3
avq.small	7085894	142.1	7085894	113.0
avq.large	7762927	156.3	7762927	121.3

Table 4.9: Comparison of net conneciton ILP/LP models

As seen in Table 4.9, the produced wirelengths of the net connection model are similar for both the ILP and the relaxed LP version. In Table 4.10, the total channel densities of the LP model are smaller than the ILP model and they are non-integral. However, the difference is small and the non-integral solution can be rounded to an integer solution. In addition, as seen in both Table 4.9 and Table 4.10, the CPU time is largely reduced by relaxing the ILP models to LP models. Therefore, in this thesis relaxed LP formulations are used to solve the global routing problem since they reduce the CPU running time without degrading the solution quality.

4.3.4 Parameter Tuning

The ILP problem and the relaxed LP problem can be solved by using different parameters within the CPLEX solver. The selection of an appropriate set of parameters has an effect on CPU time used by the solver. For the global routing

Circuit	ILP formulation		LP Relaxation	
	tracks	Time(s)	tracks	Time(s)
prim1	174	2.14	173.2	0.4
ind1	450	6.60	448.3	0.4
bio	1082	21.2	1080.4	10.7
prim2	582	28.3	581.1	16.3
ind2	1426	72.9	1424.8	33.3
ind3	2553	278.2	2548.9	152.5
avq.small	1358	345.1	1355.7	213.0
avq.large	1687	353.7	1685.2	222.1

Table 4.10: Comparison of channel density ILP/LP models

problem, the Mixed Integer Optimizer is used for the ILP formulation. Whereas the Dual Simplex Optimizer, the Primal Simplex Optimizer and Barrier Optimizer are used for the LP formulation.

4.3.4.1 Mixed Integer Optimizer

The Mixed Integer Optimizer can be used to optimize the ILP global routing problem which contains integer variables. This optimizer solves a series of subproblems by exploiting a *branch* & *cut* algorithm. To manage these subproblems efficiently, ILOG CPLEX builds a tree in which each subproblem is a node. The root of the tree is an LP relaxation of the original ILP problem.

The Mixed Integer Optimizer will find *cuts* when the solution has one or more fractional variables. *Cuts* are constraints that cut away areas of the feasible region of the relaxation containing fractional solutions. If one or more fractional variables still exist after cuts are added, CPLEX branches on a fractional variable to generate two new subproblems, each with more restrictive bounds on the branching variable.

The Mixed Integer Optimizer repeats the process until an all-integer solution is found.

Several parameters have to be adjusted to find a feasible solution:

- **Cutoff:** CPLEX cuts off nodes when the value of the objective function associated with the subproblem at that node is worse than the cutoff value. The $lowercutoff$ (when maximizing the objective function) is set to 100000 in this thesis.

- **Emphasis:** This parameter specifies whether CPLEX should emphasize feasibility or optimality as it solve the problem. If emphasis is set to "0", CPLEX tends to find a proved optimal solution with less considering the processing time. Otherwise, with a setting of "1", the first feasible but not a proved optimal solution is found.

- **Backtrack:** If the optimizer emphasis on proving optimality, backtrack is set to find feasible solutions more frequently. It is set to "1" in this thesis.

- **Node Select Strategy:** This parameter sets the rule for selecting the next node to process when backtracking. It is set to "0" to choose the most recently created node; set to "1" to choose the node with the best objective function for the associated LP relaxation; set to "2" to select the best estimate of the integer objective value that would be obtained from a node once all integer in-feasibilities are removed. In this thesis, node selection strategy is set to "2".

- **Variable Selection Strategy:** This parameter sets the rule for selecting the branching variable at the node which has been selected for branching. The maximum in-feasibility rule chooses the variable with the largest fractional value while the minimum in-feasibility rule chooses the variable with the smallest fractional rule. It can set to values as follows: (i) value "-1" leads more quickly to a first integer feasible solution; (ii) value "0" selects the best rule based on the problem and its progress; (iii) value "1" forces larger changes in the tree; (iv) value "2" is derived for pseudo-shadow prices; (v) value "3" is a computationally less-intensive form of pseudo costs. It is set to "2" in this thesis.

- **Starting Algorithm:** Determines the algorithm to be used to solve the initial relaxation of the ILP problem. It can be set to values as follows: (i) set to "1" to select Primal Simplex; (ii) set to "2" to select Dual Simplex; (iii) set to "3" to select Network optimizer; (iv) set to "4" to select Barrier with crossover (v) set to "5" to select Barrier without crossover.

4.3.4.2 Optimizers for LP Problem

ILOG CPLEX offers several different optimizers for LP problems, they are listed as follows:

- **Dual Simplex Optimizer:** A linear programming problem can be stated in primal or dual form, if the dual problem has an optimal solution, it has a direct relationship to an optimal solution of the primal model. CPLEX Dual Simplex Optimizer uses this relationship to produce an optimal solution of the given LP problem. Dual simplex method is the default method for

CPLEX to solve an LP problem.

- **Primal Simplex Optimizer:** Using Primal Simplex Optimizer is also an effective way to solve linear programming problems. But primal simplex method has been surpassed by dual simplex method at optimizing a linear programming due to recent advances in the dual simplex method. However, this optimizer is still used, since it works better on some certain problems where the number of variables are much larger than the number of constraints. It can also be used to give solutions to those problems which can not be solved by dual simplex method.

- **Primal-Dual Barrier Optimizer:** Primal-Dual Barrier Optimizer works efficiently in large and sparse problems (for example, more than 1000 rows or columns, relatively few nonzeros per column). It exploits a primal-dual logarithmic barrier algorithm to generate a sequence of strictly positive solutions to a problem.

4.3.4.3 Experimental Results

The ILP formulations (equation 4.4 and equation 4.10) and their relaxed versions (equation 4.12 and equation 4.13) contain a larger number of constraints than variables, simplex based algorithms are slow in solving such problems(*skewed problems*). Compared with Vannelli's work [Vann91], which adopts interior point method, ILP-GR uses the built-in optimizers of CPLEX to automatically solve ILP problems. It is easy to implement and also produces promising solutions. In this thesis, two CPLEX optimizers: the Dual Simplex Optimizer and the Primal-Dual

Barrier Optimizer are utilized to solve relaxed LP models.

Circuit	Dual Simplex	Primal-Dual Barrier
	time(sec)	time(sec)
prim1	0.37	0.37
ind1	0.41	0.40
prim2	0.51	0.78
bio	7.64	6.51
ind2	28.1	21.6
ind3	57.9	56.3
avq.small	128.1	113.0
avq.large	131.5	121.3

Table 4.11: Comparison of Dual Simplex and Primal-Dual Barrier Optimizers

In Table 4.11, these two optimizers are compared in terms of the running time (use the net connection model). The Primal-Dual Barrier Optimizer is faster on most circuits especially the larger circuits. From the experimental results, it can be concluded that Primal-Dual Barrier optimizer is suitable to solve *skewed problems*.

Circuit	SHGR	ILP-GR	Improvement(%)
fract	21822	21187	2.9 %
struct	448438	434832	3.0 %
prim1	891798	830365	6.9 %
ind1	1650600	1509809	8.5 %
prim2	4201487	4151385	1.2 %
bio	6327872	5266555	16.7 %
ind2	50216723	45032892	10.3 %
ind3	93136107	92599062	0.6 %
avq.s	7739168	7085894	8.4 %
avq.l	8477116	7762927	8.4 %
average	17311113	16469490	6.7 %

Table 4.12: Wirelength comparison between SHGR and ILP-GR

Table 4.12 and Figure 4.18 compare SHGR with ILP-GR (use the net connection model) in terms of the produced wirelength. As seen in Table 4.12, ILP-GR achieves an improvement by 6.7% over the wirelength yielded by SHGR. It proves that by constructing MSRTs to connect nets and being independent on the net ordering, the ILP based approach can improve the solution qualities of the global routing problem. ILP-GR provides the optimal bound for other global routers evaluating their own solutions.

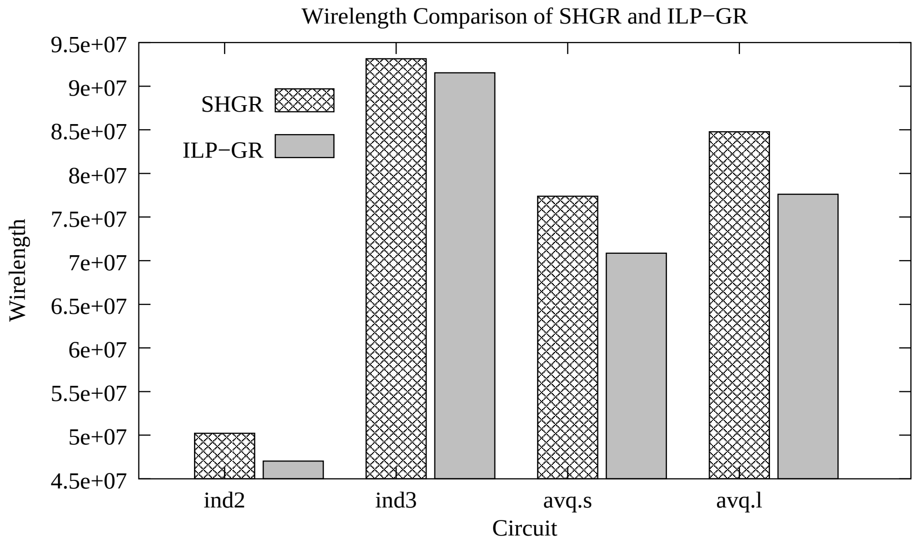


Figure 4.18: Wirelength comparison of SHGR and ILP-GR

However, in order to obtain shorter wirelength ILP-GR consumes longer CPU time to construct MRSTs and solve formulated ILP problems. As seen in Table 4.13 and Figure 4.19, ILP-GR is compared with SHGR in terms of the CPU time. Obviously, ILP-GR increases the CPU time by an average of 26.8%, since MRST construction and ILP problem are both time-consuming processes. This limits the application of ILP-GR on large sized circuits. Therefore, as circuits increase in size, the selection of ILP-GR should be avoided.

Circuit	SHGR	ILP-GR	Improvement(%)
fract	0.1	0.3	-66.7 %
struct	2.2	5.7	-61.4 %
prim1	4.5	9.4	-52.1 %
ind1	130	169	-23.1 %
prim2	150	198	-10.7 %
bio	326	489	-33.3 %
ind2	1411	2192	-35.6 %
ind3	1734	3094	-56.1 %
avq.s	12679	15289	-17.1 %
avq.l	12639	15444	-18.2 %
average	2908	3689	-26.8 %

Table 4.13: CPU time comparison of SHGR and ILP-GR

4.4 Summary

In this chapter, a Sequential Heuristic Global Router (SHGR), a Hierarchical Heuristic Global Router (HHGR) and an Integer Linear Programming (ILP) based Global Router (ILP-GR) were designed and implemented.

The main objective of SHGR was to decompose a multi-point wire into two-point segments using a Minimum Spanning Tree (MST) algorithm and find the minimum cost paths by enumerating a set of possible 2-bend routes. The standard cell layout features are considered in this global router. It takes advantage of the electrical equivalent pins and channels to find the optimal routes. HHGR routes the nets at different hierarchical levels and can be considered as a pre-processing model that provides good routing estimations in reasonable time. The ILP-GR considers all nets concurrently so as to avoid the net-ordering problem and constructs a set of Minimal Rectilinear Steiner Trees (MRST) for each net. It formulates the global routing problem as two Integer Linear Programming (ILP) models: net connection

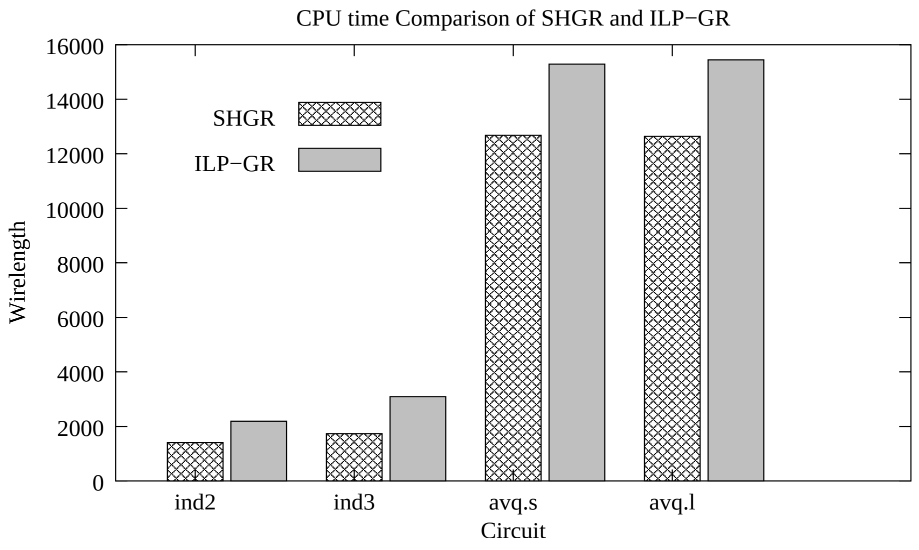


Figure 4.19: CPU time comparison of SHGR and ILP-GR

ILP model and channel density ILP model. These ILP models were further relaxed as Linear Programming (LP) models to reduce computation complexity. To speed up the running time and achieve better performance, parallel implementations of SHGR and ILP-GR routers are introduced in the next chapter.

Chapter 5

Parallel Implementation

The Sequential Heuristic Global Router (SHGR) and the ILP-based Global Router (ILP-GR) developed and introduced in chapter 4 are capable to find good solutions efficiently. However, as the complexity of VLSI circuit increases, efficiency limitations will exist. Therefore, developing parallel algorithms to solve large-size global routing problems is a necessity.

Parallelization of SHGR and ILP-GR approaches will be introduced in this chapter. The parallel algorithms are implemented by using the Message Passing Interface (MPI) technique. The MPI is a collection of message-passing libraries designed to enable distributed processing and is available for a wide range of architectures. MPI is further explained in Appendix C.

The rest of this chapter is organized as follows: Section 5.1 presents the parallel implementation of SHGR. It discusses the parallel programming paradigms and wire assignment strategy. Section 5.2 proposes the parallel implementation of ILP-GR. This is followed with experimental results and analysis in section 5.3. Finally, section 5.4 summarizes the topics discussed in this chapter.

5.1 Parallel Implementation of SHGR

The implementation of parallel algorithms varies depending on the parallel hardware architecture. Two possible parallel programming approaches have been investigated in this thesis to parallelize the Sequential Heuristic Global Router (SHGR):

1. Shared memory implementation: This parallel approach is naturally derived from the sequential algorithm implementation. The cost array is placed in a global shared memory that can be accessed by all processors. Since information is shared among processors, the wire assignment strategy is simplified. Wires are assigned to each processor using a distributed loop, in which the processors are repeatedly assigned wires for routing. In this model of computation, there is no need for communicating the updated cost array information, thus reducing the implementation complexity and the network traffic. However, the wire assignment strategy does not consider taking advantage of locality to further reduce the network traffic.

2. Distributed memory implementation: In this parallel mode, the cost array is not globally shared. Each processor has its own copy of the cost array that can not be accessed by other processors. Normally, the cost array is divided into several regions where each is assigned to an appropriate processor. The wire assignment strategy considers the locality factor (wires are assigned to these processors according to their localities). Following the routing of wires, processors broadcast their local changes to other processors and an update is

```
Parallel Heuristic Algorithm
1. Start
2. Initialization;
3. Read Circuit information and build cost array;
4. Partition cost array into regions;
5. If (wirelength < $ThresholdCost$)
     Assign wire according to the upmost pin;
6. Else
     Hold wire into the wire waiting queue;
7. End If
8. Forall processors
9.     While (not done)
10.        While (current wire queue not empty)
               Pick up a wire and route it;
               Compute the delta array;
               Send/Receive updated information;
               Update cost array;
11.        End While
12.        If
               Pick up a wire from the wire waiting queue;
13.        End If
14.    end While
15. End Forall
16. End
```

Figure 5.1: Parallel heuristic algorithm

made to each copy of the cost array. In this mode of operation, waiting time caused by blocking is avoided since processors work independently. However, communication between the processors tends to increase rapidly. The outline of this parallel routing heuristic is illustrated in Figure 5.1. Its complexity is $O(W)$, where the W is the number of wires.

Since it is easy to scale parallel algorithms from smaller number of processors to larger number of processors, also due to the availability of a distributed network of

SUN Sparc stations, a distributed memory parallel mode is adopted to parallelize the sequential global router in this thesis.

5.1.1 Distribution of Cost Array

As stated previously, in distributed parallel architectures, all processors access their own copies of the cost array. Therefore the cost array is usually divided into several approximately equal-sized regions. One possible cost array distribution strategy is to assign a region to every processor and each being responsible for routing the wires within the designated region. When a wire extends to a different region, the task is passed to the processor which owns that region. This implementation may lead to load unbalance if several wires were located in a single region of the cost array. Furthermore, it is likely that several wires will extend to other regions which might cause a huge communication bottleneck.

To reduce the network traffic, an alternative cost array distribution strategy is proposed in this thesis. The cost array is divided into a number of approximately equal-sized regions. The division considers the standard cell row based layout feature. Each region may consist of several rows and channels and is owned by a specific processor (i.e. *region-owner-proc*). Each *region-owner-proc* has a global view of the whole array as illustrated in Figure 5.2. A new data structure "*delta array*" is added to keep track of the changes of the cost array. This strategy can effectively reduce the network traffic which is caused by passing the extended wires tasks. The detailed wire assignment and data update strategy are discussed next.

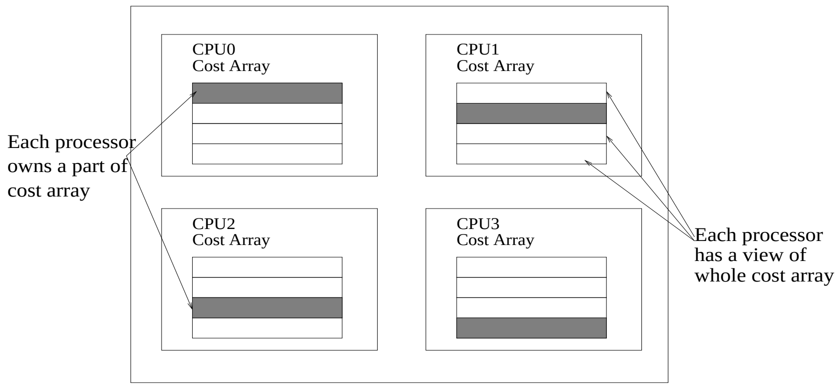

CPU0
Cost Array
CPU1
Cost Array
CPU2
Cost Array
CPU3
Cost Array
Each processor owns a part of cost array
Each processor has a view of whole cost array

Figure 5.2: Distribution of the Cost Array

5.1.2 Wire Assignment Strategy

The wire assignment strategy takes locality into account as its main feature. The first goal of the assignment strategy is to determine the region that a wire belongs to. The region where the up-most pin of the wire resides is treated as the principle region. Next, the wire is assigned to the *region-owner-proc* which owns the region. As illustrated in Figure 5.3, the up-most pin of $wire_1$ resides in $region_0$. Thus $wire_1$ belongs to $region_0$, and therefore assigned to the *region-owner-proc* CPU_0.

In most cases, wires assigned to a processor are likely to extend into regions owned by other processors. This will initiate the requirements of the cost array updates between processors. The sender processor (i.e. the *region-owner-proc*) must broadcast routing information to related receiver processors. As illustrated in Figure 5.3, $wire_1$ is owned by CPU_0, but extends to the regions owned by CPU_2 and CPU_3. The routing task is performed by the *region-owner-proc* CPU_0. When CPU_0 routes the wire outside its owned region, CPU_2 and CPU_3 are notified that certain channels and feedthroughs are utilized in their owned regions.

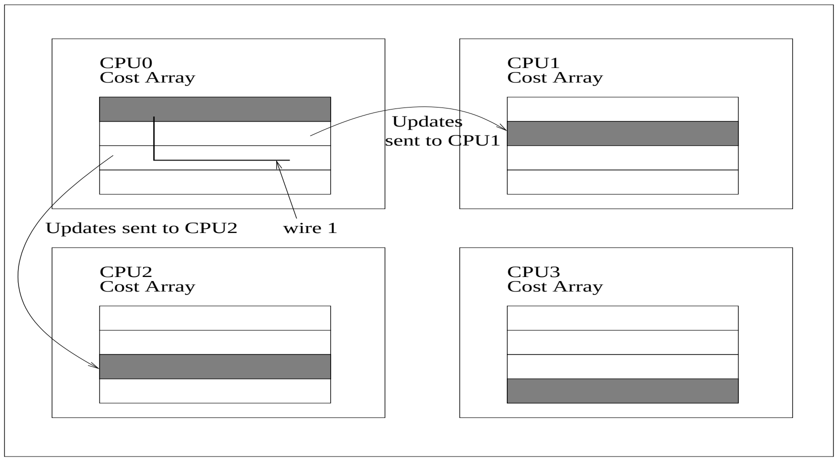


Figure 5.3: Wire assignment strategy

In order to balance the number of wires assigned to each processor, a threshold parameter "*ThresholdCost*" is introduced whose value is based on the length of wires. A wire is assigned to the *region-owner-proc* according to its up-most pin location if the wire length is lower than "*ThresholdCost*". Otherwise, it will be put into a queue and stored until the final step of the wire assignment. In the final step, the pending wires are assigned to the processors to even the balance load ignoring locality.

5.1.3 Cost Array Update

Since each processor has its own local distributed memory, it can only directly access positions in its copy of the cost array. In order to realize changes made to the cost array by other processors, communications and synchronizations must be initiated. A method that can be used to perform the cost array updates is to pass the information to other processors immediately after each wire is routed. The update packet contains coordinates of the start and end points of each segment of the wire as well as its wire length. However, sending each wire updating information can generate a large amount of network traffic. For very large circuits (which contain a large number of wires) updating information after routing each wire is quite costly.

In order to reduce the network traffic and track the changes, a new data structure "*delta array*" is added. The "*delta array*" is used to notify other processors of the changes made to the cost array between updates. It has the exact same dimensions as the cost array. Following the routing of wires, the "*delta array*" is computed to track the changes made to a couple of wires. The sender processor checks the "*delta array*" to construct an update packet which is a bounding box that contains all the changes within a certain region and the coordinates of the bounding box.

5.2 Parallelization of ILP-GR

As described in chapter 4, ILP-GR produces a set of possible Minimal Rectilinear Steiner trees (MRST) to connect the terminals of each net. However, considering all possible trees for a net results in a large number of variables and constraints which complicates solving the optimization problem. Moreover, the Steiner tree generating algorithm is computationally expensive and time consuming [Gare79]. Therefore, by taking advantage of distributed architectures, a parallel version of ILP-GR is implemented to meet this challenge.

5.2.1 Parallel Implementation

The parallel ILP based algorithm takes the distributed memory Single Instruction Multiple Data (SIMD) parallel model. Since no parallel version of the CPLEX solver is available, only a subset of ILP-GR (Steiner Tree generation task) is parallelized in this thesis.

As illustrated in Figure 5.4, the algorithm consists of three phases:

1. Phase t_1 (initialization phase): All processors from CPU_0 to CPU_n read the netlist information from the placement CAD tool. The complete netlist is partitioned into sub-lists and assigned to related processors.

2. Phase t_2: Each processor is assigned a subtask from the previous phase t_1. For all the processors except CPU_0, after all the possible MRST sets are produced, the data is sent to CPU_0. CPU_0 receives results and combines them into a single entity.

3. Phase t_3: The combined data from phase $t2$ is used to formulate the global routing problem as an ILP problem. The ILP problem is relaxed and solved by the CPLEX solver.

The parallel implementation of phase $t1$ and $t2$ is illustrated in Figure 5.5.

5.2.2 Wire Assignment Strategy

Provided there are N nets in a circuit, the number of pins associated with net i is n_i, and each net has a weight value associated with it. In order to partition the nets and distribute them among the processors, the nets are first sorted according to

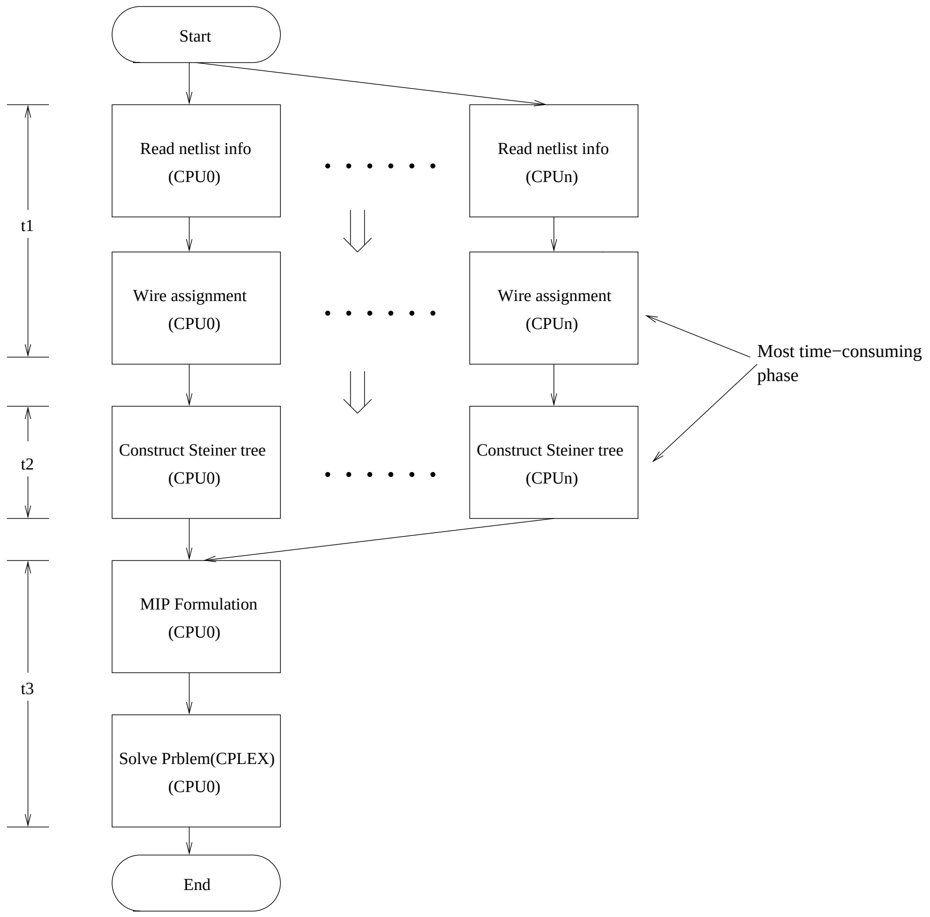


Figure 5.4: The parallel implementation in the whole global routing process

their weights. Given p processors, the nets are assigned to each processor until the number of pins is larger than the average number $\sum_1^N n_i/p$. There are two possible wire or net assignment strategies for the ILP based global router.

1. Center partition: The pins of a net may be located anywhere among the rows of a circuit. Usually the pins located at the same rows will be connected by putting tracks within channels. Since nets that stay around the same rows tend to put tracks into the same channel, there will be more runtime

```
ILP Parallel Algorithm
1. **Start**
2. All processors read circuit information;
3. Assign nets to each processor;
4. **If** (rank==0)
     Processor $CPU_0$ generates Steiner Trees;
     Update Steiner tree set;
     Processor $CPU_0$ receives tree data from all other processors;
5. **End If**
6. **If** (rank > 0)
     Processor $CPU_i$ generates Steiner Trees;
     Update Steiner tree set;
     Processor $CPU_i$ sends tree data to Processor $CPU_0$;
7. **End If**
8. **End**
```

Figure 5.5: ILP based parallel algorithm

interaction between those vertically close nets. Therefore, the net weight is calculated based on this relationship. Assume the number of pins on net i is n_i and the position of a pin is (x_j, y_j), where $j = 1, ..., n_i$. The center of the net is given by: $(\sum_1^{n_i} x_j/n_i, \sum_1^{n_i} y_j/n_i)$. Since the execution time on finding vertical routes tends to be longer, the weight w of the net can be derived from the y coordinate of its center, where $w = \sum_1^{n_i} y_j/n_i$. As seen in Figure 5.6, net i has four pins: p_1 $(1,0)$, p_2 $(2,0)$, p_3 $(3,2)$ and p_4 $(2,2)$. Its center is calculated as $(2,1)$ and its weight is 1.

2. Pin number weight partition: This strategy sorts and assigns the nets according to the number of pins on the net such that larger nets are assigned more weights. Assume the number of pins on net i is n_i, a weight $-n_i^\alpha$ is given to this net, where α is a positive number. Experimental results show that a value for α within the range of 1.5 is adequate. Following the sorting stage,

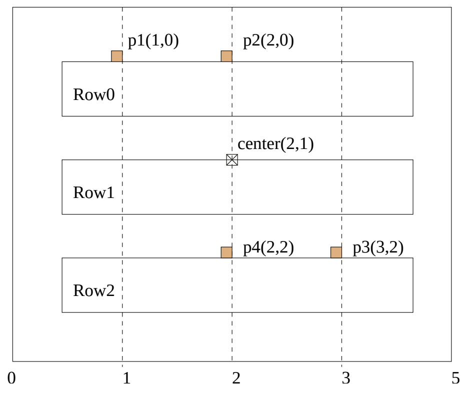


Figure 5.6: Center partition wire assignment

the nets are evenly distributed to each processor in a round-robin manner.

The center partition strategy attempts to balance the work loads by only considering the assignment of equal-sized load of pins to each processor. However, constructing the MRST has the largest computation complexity in the global routing algorithm. Even if each processor works on the same number of pins, the work load may not be balanced since some processors are assigned large nets. For example, both CPU_0 and CPU_1 are assigned 10 pins. CPU_0 has 5 two-pin nets, CPU_1 has 1 two-pin net and 1 eight-pin net. Even though the number of pins is balanced among the two processors, the work load of CPU_1 is much larger because constructing MRST for the eight-pin net becomes the dominant part of the computation. In order to balance the work load efficiently, the pin number weight partition strategy is used in this thesis.

5.3 Experimental Results

The parallelization of SHGR and ILP-GR shows good solution quality. Table 5.1 and Table 5.2 list speedups obtained by parallelizing the two global routers on 2, 4 and 6 processors (can easily scale up to more processors).

Circuit	2 procs	4 procs	6 procs
prim1	1.94	3.27	3.60
ind1	1.91	3.71	5.20
bio	1.95	3.87	5.28
prim2	1.89	3.85	5.16
ind2	1.94	3.89	5.49
ind3	1.98	3.79	5.31
avq.small	1.82	3.72	5.24
avq.large	1.77	3.65	5.11

Table 5.1: Speedup of the parallel heuristic router

Circuit	2 procs	4 procs	6 procs
prim1	1.95	3.29	3.60
ind1	1.91	3.68	5.21
bio	1.90	3.87	5.22
prim2	1.90	3.80	5.11
ind2	1.84	3.79	5.29
ind3	1.91	3.81	5.14
avq.small	1.87	3.65	5.01
avq.large	1.72	3.61	5.08

Table 5.2: Speedup of the ILP based router

In Figure 5.7 and 5.8, the plots of speedups on several medium and large circuits (*prim*2, *ind*3, *avq.small* and *avq.large*) are given. The x-axis is the number of processors utilized, the y-axis is the value of speedup.

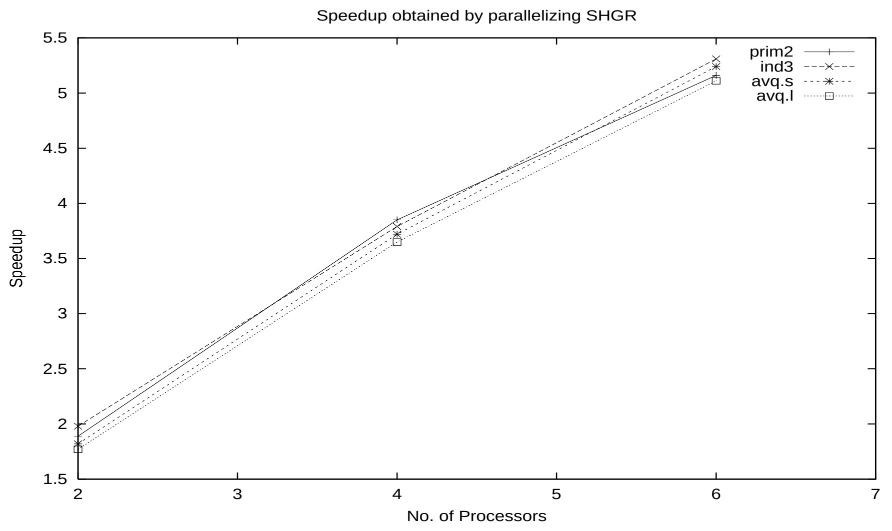


Figure 5.7: Speedup obtained by parallelizing SHGR on large benchmark circuits

According to *Amdahl's law* (discussed in Section 2.4.3), ideal linear speedups are impossible due to the existence of serial portions within programs. As seen from Figure 5.7 and 5.8, almost linear speedups are achieved by the parallel implementations on both global routers.

Circuit	Total Channel Density		
	Before Parallel	After Parallel	Improve %
ind1	450	439	2.44%
prim2	582	573	1.54%
bio	1082	1071	1.01%
ind2	1426	1412	0.98%
ind3	2553	2534	0.74%
avq.small	1358	1341	1.20%
avq.large	1687	1660	1.59%
average	1305	1290	1.36%

Table 5.3: Parallelism improves the total density

In addition, by exploiting locality in the task assignment, the parallel imple-

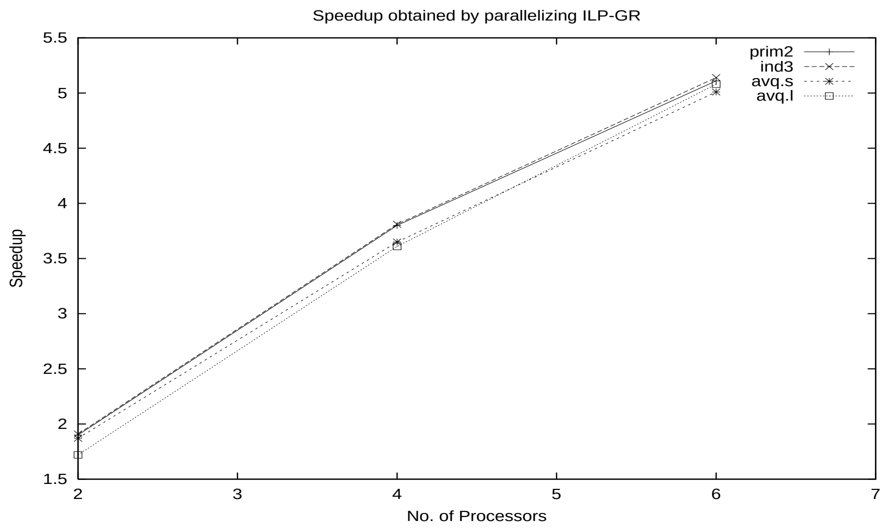


Figure 5.8: Speedup obtained by parallelizing ILP-GR on large benchmark circuits

mentations also improve the total density performance which in place leads to a chip area reduction. Table 5.3 shows that the total density is reduced on average by 1.59% when parallelizing the heuristic global router.

From the discussion above, the parallel implementations can be used to solve very large sized global routing problems, where the traditional serial algorithms meet the timing limitation. Moreover, due to the architecture of distributed memory parallel mode, they are easy to scale from 6 processors to run on more processors such that the CPU time is further shortened.

5.4 Summary

In this chapter, two parallel algorithms were implemented achieving good speedups and solution quality. Due to the fact that a network of SUN Sparc stations was available, the distributed memory parallel mode was adopted for both algorithms. With respect to the Sequential Heuristic Global Router (SHGR), the wire assignment and updating strategy were discussed. A new parameter "$ThresholdCost$" and data structure "*delta array*" were introduced to balance the work load and reduce the network traffic. Two wire assignment strategies for the ILP-based Global Router (ILP-GR) were introduced in the form of: center partition and pin number weight partition. In this thesis, the pin number weight partition strategy was utilized due to its capability of balancing the work load.

Chapter 6

Conclusions and Future Directions

One objective of global routing is to determine interconnections between cells and modules without violating resource constraints. As discussed in chapter 1, with the technology moving to sub-micron rules the interconnection delay and capacitance have become dominant factors in determining the circuit speed (delay) and power consumption respectively. A reduction in the interconnection delay and power consumption can be achieved by producing shorter wirelength. Yet another objective of global routing is to minimize the chip area which can be achieved by a reduction in channel density. Since the produced wirelength and channel density have important impacts on the global routing solutions, they are used to evaluate the efficiency of implementations in this research.

6.1 Conclusions

In this thesis several global routers were proposed in the form of: a Sequential Heuristic Global Router (SHGR), a Hierarchical Heuristic Global Router (HHGR) and an ILP-based Global Router (ILP-GR). These routers were able to achieve high quality solutions for the standard cell design style. To achieve better quality and further reduce the running time parallel versions of SHGR and ILP-GR were implemented. The contributions of each proposed global router and the parallel implementations are summarized as follows:

- SHGR searches for the minimum cost path of each net by enumerating a set of possible 2-bend routes. It is an efficient implementation that can be used for large sized VLSI design circuits. On average a 16% improvement was achieved over wirelength produced by placement based on HPWL. These results are on average 6.7% away from optimal solutions produced by ILP models, which is partially due to the net-ordering problem. In addition, SHGR constructs MSTs to connect pins, which can not produce the shortest wirelength. However, the complexity to generate MSTs is linear, which enables SHGR to consume less CPU time for large sized global routing problems.

- HHGR is a novel global routing technique which routes circuits at different hierarchical levels. Compared with other previous hierarchical global routers, such as [Burs83] [Heis91] and [Kuh86], it is based on a multiple-clustering technique to build the hierarchical levels and can produce a good estimation solution in short time. In most cases, it is necessary to predict the routability for a VLSI design before the global routing phase starts. Therefore HHGR has important practical values which can be used as a pre-processing global routing model. Experimental results show that HHGR consumes on average 89% less CUP time than SHGR.

- ILP-GR formulates the global routing problem as two Integer Linear Programming (ILP) models. It overcomes the net-ordering problem by routing net simultaneously and also improves the solution quality by investigating the effective generation of MRSTs. ILP-GR provides the optimal solution bound which can be compared with other implementations. On average a 6.7% improvement is achieved over the wirelength solution produced by SHGR. However it consumes more CPU time which limits its use for large sized problems. The selection of ILP-GR should be avoided as circuits increase in size.

- Parallel versions of SHGR and ILP-GR are also implemented in this dissertation. As the complexity of VLSI design increases, developing parallel algorithms to solve large sized global routing problems becomes necessary. The distributed memory model is adopted in this thesis since the developed parallel algorithms can be easily scaled to run on a large number of processors. Solutions obtained indicate promising results where near linear speedups and further improvement (on average 1.36%) over channel density can be achieved. Therefore, parallel implementations can be easily adopted when serial implementations fail to obtain solutions in reasonable time.

6.2 Future Work

The two mathematical models of ILP-GR have different objectives, where the net connection formulation is to produce the shortest wirelength and the channel density model seeks to generate the smallest channel density. In the future, the two models can be combined together as a single multi-objective ILP model. The combined ILP formulation targets at finding optimal solutions that seek to shorten the wirelength and minimize the channel density simultaneously.

Future work should also include improving the Integer Linear Programming (ILP) model by considering congestion minimization. A congested area has a large number of nets passing through it, and may result in no feasible global routing solution. The ILP based approach has a global view of the problem and can be utilized to predict congested areas. The congestion estimation method should obtain a priori information about routing areas of the circuit. This information is attached to those trees which connect the nets passing through the congested area. The additional trees for these nets will pass through routing areas which are potentially less congested, thus, reducing the possibility of congestion.

Another interesting direction for future work involves improving the global routing model by incorporating timing information. With the advance in VLSI fabrication technology, deep submicron design has become more and more popular, in which the interconnection delay has produced a very huge impact on the circuit speed. In many cases, the interconnection delay can even consume from 50% to 70% of the clock cycle [Bako90]. Therefore, in order to improve the circuit performance, critical paths have to be seriously considered in future models.

Appendix A

Glossary

BFS	:	Breadth First Search
CAD	:	Computer Aided Design
CMOS	:	Complementary Metal Oxide Semiconductor
DA	:	Design Automation
FPGA	:	Field Programmable Gate Array
HHGR	:	Hierarchical Heuristic Global Router
ILP	:	Integer Linear Programming
ILP-GR	:	ILP-based Global Router
LP	:	Linear Programming
MCM	:	Multiple Chip Module
MCNC	:	Microelectronics Center of North Carolina
MIMD	:	Multiple-Instruction Multiple-Data
MPI	:	Message Passing Interface
MRST	:	Minimum Rectilinear Steiner Tree

MST	:	Minimum Spanning Tree
NP-hard	:	Non Deterministic Polynomial Hard
NP-complete	:	Non Deterministic Polynomial Complete
SHGR	:	Sequential Heuristic Global Router
SIMD	:	Single-Instruction Multiple-Data
SPMD	:	Single-Program Multiple-Data
VHDL	:	Very High Speed Integrated Circuit Hardware Description Language
VLSI	:	Very Large Scale Integration

Appendix B

CPLEX Solver

B.1 Introduction to CPLEX

ILOG CPLEX is a tool for solving linear programming (LP) problems, is defined as:

$$Maximize\ (or\ Minimize) \qquad c_1x_1 + c_2x_2 + \ldots + c_nx_n \qquad \text{(B.1)}$$

$$\begin{aligned}
subject\ to \qquad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \ \tilde{}\ b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \ \tilde{}\ b_2 \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \ \tilde{}\ b_m \\
with\ bounds \qquad & l_i \leq x_i \leq u_i \\
& l_n \leq x_n \leq u_n
\end{aligned}$$

where ˜ can be $\leq$, $\geq$, or $=$, and the upper bounds u_i and lower bounds l_i may be positive infinity, negative infinity, or any real number.

The data provided as input to this LP problem is:

$$\begin{array}{rl} Objective function coefficients & c_1, \ldots c_n \\ Constraint coefficients & a_{11}, \ldots a_{1n} \\ Right-hand sides & b_1, \ldots b_n \\ Upper bounds & u_1 \ldots u_n \\ Lower bounds & l_1 \ldots l_n \end{array}$$

The optimal solution that CPLEX returns is:

$$Variables \qquad x_1, \ldots x_n$$

B.2 Using CPLEX to Solve ILP/LP Problems

ILOG CPLEX consists of C and C++ libraries to solve linear programming and related problems, it comes in three forms:

- **CPLEX Interactive Optimizer**: An executable program that can read problem interactively or from input files which has standard formats, solve the problem, and give the solution interactively or write the solution to output files.

- **Concert Technology**: A set of C++ libraries which enables the programmer to embed CPLEX optimizer in applications written in C++.

- **CPLEX Callable Library**: A set Libraries that can be embedded into applications written in C, Visual Basic and Java to solve the LP related problems.

Because the global routing application is written in C, in this paper I will describe how to incorporate the CPLEX Callable Library into C programs. By embedding CPLEX Callable Library, developers can write applications to optimize, modify and interpret the results of linear programming (LP) problem, mixed integer programming (MIP) problem and other related problems.

In order to use the Callable Library, we need first initialize the ILOG CPLEX environment, open the problem object and fill it with data, then certain optimizers are selected to optimize the problems, at last, the application frees the problem object and releases the ILOG CPLEX environment. The detailed explanation is give as follows:

- **Initialize the CPLEX Environment** Before we use the Callable Library in the application, we need initialize certain data structures which are required by ILOG CPLEX. To initialize the CPLEX environment, the routine CPX-openCPLEX() is used, it checks ILOG CPLEX license and then return a C pointer to the ILOG CPLEX environment, the application then passed this pointer to all the other CPLEX routines to be used.

- **Open the Problem Object** After initializing the ILOG CPLEX environment, we need open a problem object by calling routine CPXcreateprob(), it returns a C pointer to the problem object, this pointer is then passed to the other CPLEX routines.

- **Fill Data in Problem Object** Once the problem object is opened, it needs to be filled with data because the original object is empty, we have to put necessary constraints, variables and coefficient matrix in it. The most used routines are CPXcopylp(), CPXnewcols(), CPXnewrows(), CPXaddcols(), CPXaddrows() and CPXchgcoeflist().

- **Optimize the Problem** The core of using CPLEX is to optimize and solve the given problem, once the problem object has been created and populated with data, we can choose appropriate optimizer to solve it. There are several different optimizer for linear programming (LP) problems and mix integer programming (MIP) problems: Dual Simplex Optimizer, Primal Simplex Optimizer, Network Optimizer and Mixed Integer Optimizer.

- **Free Problem and Release the CPLEX Environment** After the optimal solution is produced, we need call routine CPXfreeprob() to destroy the problem object and call routine CPXcloseCPLEX() to release the ILOG CPLEX environment.

Appendix C

MPI

C.1 Introduction to MPI

Message Passing Interface (MPI) is a set libraries providing message passing support for parallel/distributed applications. It is developed by an open forum (www.mpi-forum.org) of vendors, users and researchers. Compared with the previous products, MPI is a standardized open technique, it has very good scalability and memory is local to process which makes debugging very easy. The goal of MPI is to develop a widely used, practical, portable and efficient standard for writing message-passing programs.

C.2 MPI Datatypes and Basic Functions

The MPI standard is intended to be used by those who want to write message passing program in Fortran77 and C. There are two kinds of datatypes in MPI:

1. Basic datatype, which are predefined corresponding to the underlying language, Table C.1 lists the MPI datatypes used in C programs.

2. Derived datatype, which are the new datatypes constructed by the users at run time. These user defined datatypes include vector, array and struct etc. Through these user defined datatype, MPI supports the communication of complex data structures such as array sections and structures containing combinations of basic datatypes.

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short
MPI_INT	signed int
MPI_LONG	singed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Table C.1: The basic MPI datatypes

C.3 MPI Core Functions

The basic communication mode of MPI is point to point communication, in which the data transmission is between a pair of processes, one side is sending and the other is receiving, the send and receive operations are the core operations in MPI. The basic send and receive functions in MPI are MPI_Send and MPI_Recv. Function MPI_Send send the data from source processor to target processor and function MPI_Recv receive the data from source processor.

Some other important core functions are:

1. MPI_Init() Must be called before any other MIP functions, perform an initiation for the MPI library.

2. MPI_Finalize() Must be called after all MIP functions, perform the clean-up for MPI library and free memory.

3. MPI_Comm_rank() Return the integer rank of the running process.

4. MPI_Comm_size() Return the integer number of processes.

A simple MPI program should at least contain these core functions, a "Hello World" example is given as follows:

```
int main( argc, argv )
int argc;
char **argv;
{
int rank, size;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
printf( "Hello world! I'm %d of %d\n",
        rank, size );
MPI_Finalize();
return 0;
}
```

This ”Hello World” example first initiate the MPI library using the function MPI_Init. next it uses function MPI_Comm_rank() and MPI_Comm_size() to obtain the total number of the processes and the integer rank of the each running process. Finally, it calls function MPI_Finalize() to clean up the MPI library.

Bibliography

[AN87] C.D. Thompson A. Ng, P. Raghavan, "A Language for Describing Rectilinear Steiner Tree Configuration," In *Proc. 23rd Design Automation Conference*, pp. 319–320, June 1987.

[Babb93] J. Babb and R. Tessier, "Vitual Wires' Overcoming Pin Limitations in FPGA-based Logic Emulators," In *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. , May 1993.

[Bako90] H.B. Bakoglu, *Circuits , Interconnections, and Packaging for VLSI*, Addison-Wesley, 1990.

[Bane94] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*, Prentice Hall, Englewood Cliffs,NJ, 1994.

[Baza77] M.S. Bazaraa and J.J. Jarvis, *Linear Programming and Network Flows*, New York, 1977.

[Behj02] Laleh Behjat, *Thesis: New Modeling and Optimization Techniques for the Global Routing Problem*, University of Waterloo, 2002.

[Blan85] J.P. Blanks, "Near Optimal Quadratic Based Placement for a Class of IC Layout Problems," *IEEE Circuits and Devices*, vol. 1, No. 6, pp. 31–37, September, 1985.

[Brou90] R.J. Brouwer and P. Banerjee, "PHIGURE: A Parallel Hierachical Global Router," In *Proc. 27th Design Automation Conf.*, pp. 360–364, Jun. 1990.

[Burs83] M. Burstein and R. Pelavin, "Hierarchical Wire Routing," In *IEEE trans. Computer-Aided Design, Vol. CAD-2*, pp. 223–234, Oct. 1983.

[Card91] R. Carden and C.K. Cheng, "A Global Router Using An Efficient Approximate Multicommodity Multiterminal Flow Algorithm," In *28th ACM/IEEE Design Automation Conf.*, pp. 316–321, 1991.

[Cho00] Jun Dong Cho, “Wiring Space and Length Estimation in Two-dimensional Arrays,” In *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Volume: 19 Issue: 5*, pp. 612 –615, May 2000.

[Cong92] J. Cong and B. Preas, “A New Algorithm for Standard Cell Global Routing,” *Integration: the VLSI Journal*, pp. 49–65, November 1992.

[Cong98] J. Cong and P. Madden, “Performance Driven Multi-Layer General Area Routing for PCB/MCM Designs,” In *Design Automation Conference*, pp. 356–361, IEEE, 1998.

[Dijk59] E. W. Dijkastra, “A note on two problems in connection with graphs,” *Numerische Mathematick*, pp. 269–271, 1959.

[Gare79] M.R. Garey and D.s. Johnson, *Computers and Intractability:A Guide to the Theory of NP-completeness*, Freeman, San Francisco,CA, 1979.

[Hana85] M. Hanan, “On Steiner’s Problem with Rectilinear Distance,” *in VLSI Circuit Layout: Theory and Design,T.C. Hu and E.S. Kuh*, eds. IEEE, pp. 133–138, 1985.

[Heis91] J. Heisterman and T. Lengauer, “The Efficient Solution of Integer Programs for Hierarchical Global Routing,” *IEEE trans on Computer-Aided Design*, pp. 748–753, Jun. 1991.

[Horg04] Jack Horgan, “Low Power Soc Design,” In *EDA weekly Review*, pp. , May 2004.

[Hu00] Jiang Hu and S.S. Sapatnekar, “A Timing-Constrained Algorithm for Simultaneous Global Routing of Multiple Nets,” In *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, pp. 99 –103, 2000.

[HU85a] T.C. HU and E. Kuh, “Theory and Concepts of Circuit Layout,” In *VLSI Circuit Layout:Theory and Design*, pp. 3–18, IEEE PRESS, New York, 1985.

[Hu85b] T.C. Hu and M. T. Shing, “A Decompsition Algorithm for Circuit Routing,” *in VLSI Circuit Layout: Theory and Design,T.C. Hu and E.S. Kuh*, eds. IEEE, pp. 144–152, 1985.

[Hwan76] F. Hwang, “On Steiner Minimal Trees with Rectilinear Trees,” *SIAM Journal of Applied Mathematics*, pp. 104–114, Jan. 1976.

[JRos88] J.Rose, “LocusRoute: A Parallel Global Router for Standard Cells,” In *Proc. of 25th DAC*, pp. 189–195, Jun. 1988.

[Kahn95] A. Kahng and G. Robins, *On Optimal Interconnects for VLSI*, Kluwer Academic, Boston, MA, 1995.

[Kang03] S.M Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design, 3rd Edition*, McGraw-Hill, 2003.

[Kary97] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partioning: Application in VLSI Design," In *Proceedings of* 35*th DAC*, pp. 526–529, ACM/IEEE, Las Vegas, Nevada, June 1997.

[KMik68] K.Mikami and K.Tabuchi, "A Computer Program for Optimal Routing of Printed Circuit Connectors," In *IFIPS Proc.H47*, pp. 1475–1478, 1968.

[Kozm91] K. Kozminski, "Benchmarks for Layout Synthesis - Evolution and Current Status," In *Proceedings of The* 28*th DAC*, pp. 265–270, IEEE/ACM, Portland, Oregon, 1991.

[Kris02] J. Lou S. Thakur S. Krishnamoorthy and H.S. Sheng, "Estimating Routing Congestion Using Probabilistic Analysis," In *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Volume: 21 NO. 1*, pp. 32–42, Jan. 2002.

[Kuh86] E.S. Kuh and M. Marek-Sadowska, *Global Routing*, in Layout Design and Verification, T. Ohtsuki, ed., Elsevier Science Publishing, New York, N.Y., 1986.

[KW88] Lee K.-W and C. Sechen, "A New Global Router for Row-based Layout," In *in Proc. Int. Conf. on Computer Aided Design*, pp. 180–183, Nov. 1988.

[Lee61] C.Y. Lee, "An Algorithm for Path Connection and Its Application," *IRE Transactions on Elctronic Computers*, pp. 346–365, 1961.

[Leng90a] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Chichester, England, 1990.

[Leng90b] T. Lengauer, *Combitational Algorithms for Integrated Circuit Layout*, J. Wiley, New York, NY, 1990.

[Meix90] G. Meixner and U. Lauther, "A new Global Router Based on a Flow Model and Linear Assignment," In *in Proc. Int. Conf. on Computer Aided Design*, pp. 44–47, 1990.

[Moor59] E.F. Moore, "The Shortest Path through a Maze," *Annals of the Harvard Computation Laboratory*, pp. 185–292, 1959.

[Pedr89] M. Pedram and H. Preas, "Interconnection Length Estimation for Optimized Standard Cell Layouts," In *in Proc. Int. Conf. on Computer Aided Design*, pp. 390–393, 1989.

[RK87] R.L. Rivest R.M. Karp, F.T. Leighton, *Global Wire Routing in Ttwo-demensional Arrays.Algorithmica, 2:113-129,1987*, 1987.

[SA01] ILOG S.A., *ILOG CPLEX 7.5 Users Manual*, ILOG S.A. R&D office, 1661 Route des Dolines, 06560 Valbonne, 2001.

[Sarr89] C. Chiang M. Sarrafzadeh and C.X. Wong, "A Powerful Global Router: Based on Steiner Min-Max Trees," In *in Proc. Int. Conf. on Computer Aided Design*, pp. 2–5, 1989.

[Sarr94] C. Chiang M. Sarrafzadeh and C.X. Wong, "A Powerful Global Router: Based on Steiner Min-Max Trees," *Trans. on CADICS*, IEEE, pp. 1461–1469, 1994.

[SB91] H. Chen S. Burman and N. Sherwani, "Improved Global Routing Using λ-Geometry," In *The Proceedings of 29th Annual Allerton Conference on Communications,Computing, and Controls*, pp. , Oct. 1991.

[Sech86] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," In *IEEE Design Automation Conf.*, pp. 432–439, 1986.

[Sech88] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Academic Press, Boston, MA, 1988.

[Shah88] F. Shahrokhi and D.W. Matula, *The Maximum Concurrent Flow Problem*, pp. , Manuscript, New Mexico Tech, Jan. 1988.

[Shah91] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," *ACM Computing Surveys*, vol. 23, No. 2, pp. 143–220, 1991.

[Sher99] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic, Boston, MA, 1999.

[SKan03] S.Kang and Y. Leblebici, *CMOS Digital Integrated Circiuts*, The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020, 2003.

[SM91] J. Hashmi S. Miriyala and N. Sherwani, "Switch Steiner Tree Problem in Presence of Obstacles," In *28th Int. Conf. on Computer Aided Design*, pp. 596–599, Nov. 1991.

[Souk78] J. Soukup, "Fast Maze Router," In *Proceedings of 15th Design Automation Conference*, pp. 100–102, IEEE, 1978.

[Sun04] Hao Sun and Shawki Areibi, "Global Routing for VLSI Standard Cell," In *Canadian Conference on Electrical and Computer Engineering*, pp. , May 2004.

[Sun90] X. Sun and L. M. Ni, *Another View on Parallel Speedup*, Michigan State University, East Lansing, MI 48824-1027, 1990.

[Swar93] William Swartz and Carl Router, "A New Generalized Row-Based Global Router," In *Proc. of ICCAD*, pp. 491–498, 1993.

[THC90] R.L. Rivest T. H.. Cormen, C. E.Leiserson and C.Stein, *Introduction to Algorithms*, The MIT Press, Five Cambridge Center, Cambridge, MA 01142, 1990.

[Ting83] B. Ting and B. Tien, "Routing Techniques for Gate Array," In *IEEE trans. Computer-Aided Design ,Vol. CAD-2*, pp. 301–312, Oct. 1983.

[Ueda86] K. Ueda, R. Kasai, and T. Sudo, "Layout Strategy, Standardization, and CAD Tools," *Layout Design and Verification*, North-Holland, pp. 1–54, 1986.

[Vann91] A. Vannelli, "An Adaptation of the Interior Point Method for Solving the Global Routing Problem," *IEEE trans on Computer-Aided Design*, pp. 193–203, February 1991.

[Yang03] Z. Yang, *Thesis: Area/Congestion-driven Placement for VLSI Circuit Layout*, University of Guelph, 2003.