Waseda University Doctoral Dissertation

Research on Combined Approaches of Graph Theory and Mathematical Programming for High-Level and Physical Synthesis of VLSI

Cong HAO

Graduate School of Information, Production and Systems

Waseda University

July 2017

Abstract

Along with explosive VLSI applications in modern technologies, VLSI design using software tools becomes of the utmost importance, which is referred to as electronic design automation (EDA). EDA design flow, also called IC design flow, is a process which consists of several automated steps to accomplish the design of an IC. Since EDA tools play an important role in developing ICs with high performance, low cost and fast time-to-market, the development of high efficiency algorithms used in EDA design flow for various optimization objectives are attracting more and more research interests. The general EDA design flow contains several representative steps including high level synthesis (HLS), logic synthesis, and physical synthesis, etc., to meet various design requirements such as power, chip area, clock frequency and signal delay.

Since mid-1970s, the investigations of EDA design methodologies have been carried on for decades, and significance achievements have been made for automated designs instead of manual designs. Along with the development of commercial EDA tools, EDA companies are also growing fast and continuously, proposing latest design solutions. Apart from the achievements, there are still unsolved problems in the flow, either because of the hardness of the problem, or because of the rapidly growing problem size. For example in HLS, for the researches in low power scheduling and binding, the solution quality is still lower than manual designs, which has a large room for improvement; for the researches in interconnection optimization, some important problems such as the port assignment problem, are ignored and not being fully investigated, which is regretful since they do has a great affection on total power consumption. Therefore, algorithms with high optimality are still expected, which equal or even excel manual designs. In addition, in physical synthesis, one of the problems is, together with the growing design and circuit size, some problems, say the TSV assignment problem, become extremely huge, which makes it difficult to be solved using existing algorithms. Therefore, algorithms with high efficiency are expected to handle large scaled problems in physical synthesis.

Motivated by the existing issues, in this research, several problems in two main stages, are to be studied, including **high level synthesis** and **physical synthesis**, because most problems in high level synthesis and physical synthesis share similar problem formulations, say Integer Linear Programming (ILP) formulation or graph formulation. Given their similar properties, in this research, some combined approaches of **graph theory** and **mathematical programming**, is to be investigated and to be applied on HLS and physical synthesis to solve the above issues. In the HLS process, several major steps including scheduling, resource allocation and binding, and interconnection optimization are discussed. In physical synthesis for 3D IC, the TSV insertion problem for 3D-IC is mainly studied. They are studied by common technologies, including graph theory, mathematical programming and iterative methods say local search with random re-start, with the principle of improving the algorithm optimality and efficiency. This thesis is organized as the follows.

Chapter 1, [**Introduction**], first gives a brief introduction to EDA design flow, mainly including high level synthesis, logic synthesis and physical synthesis. Some existing problems in current EDA design flow are addressed, for example in high level synthesis, the optimality of existing algorithms is not high enough, and in physical synthesis the algorithm efficiency is low especially for large scaled problems. Motivated by the existing problems, the principles of this research are given, which are improving algorithm optimality for small and medium sized problems, as well as improving algorithm efficiency for large sized problems. Then, three major topics of this research are introduced, including: (1) the multiple supply/threshold voltage scheduling for dynamic/leakage power minimization, which corresponds to the scheduling step in HLS; (3) the interconnection optimization between functional units and registers, which corresponds to the interconnection allocation step in HLS; and (3) the through silicon via (TSV) insertion on 3D-ICs to reduce routing wire length, which corresponds to the floor plan and placement step in physical synthesis. Their common technologies are also briefly addressed. Finally, the background knowledge of HLS, 3D IC and TSV insertion problem are briefly addressed.

Chapter 2, [**A Unified Scheduling Approach with Multiple V_{dd} or/and V_{th} in HLS**], discusses the dynamic and leakage power minimization problem using multiple V_{dd} and V_{th} technology in operation scheduling. The combined scheduling and binding method is also discussed in this chapter. For this problem, the inputs are small or medium scaled, but the optimality of existing algorithms are still not high enough because of the hardness of this problem. Therefore, the purpose of this topic is to propose algorithms with high optimality, which equal or even excel the manual designs.

In this chapter, a unified scheduling approach which is applicable to various optimization problems is proposed, including: (1) dynamic power and resource usage co-optimization; (2) leakage power optimization; and (3) dynamic power and leakage power co-optimization. To deal with different objectives with high flexibility, three problems are divided into two common sub-problems including delay assignment and resource density variance minimization. Then a vertex potential based mobility allocation model is proposed to solve two sub-problems simultaneously. On the proposed mobility graph, the network simplex method is applied to solve the mobility allocation problem, which optimizes both dynamic power and resource usage by adjusting the vertex potentials. The mobility allocation is iteratively updated by local search until the algorithm meets stop criteria. The combined scheduling and binding for power minimization is also investigate, which conducts binding after scheduling only when the scheduling results are promising in reducing current overall power consumption.

Experimental results show that, for dynamic power and resource co-optimization, the proposed unified scheduling approach produces optimum solutions for all 6 benchmarks with 15 groups of data; for leakage power optimization, it also greatly excels the latest existing work, by 20% leakage power reduction and 52 times speedup. Besides, for dynamic power and leakage power co-optimization, the Pareto Solutions are studied.

Chapter 3, [**Interconnection Allocation Between Functional Units and Registers in HLS**], discusses the interconnection optimization techniques in HLS, which has not been fully investigated. Algorithms are proposed for the port assignment problem between functional units and registers. This problem is also small or medium scaled, but has a large impact on chip design, such as chip area, power

consumption and signal delay. Further, this problem may be solved iteratively in HLS tools, which requires high efficiency of the algorithms.

In this chapter, the port assignment problem for binary commutative operators for interconnection complexity reduction is discussed. First, the port assignment problem is formulated on a constraint graph. By constructing a spanning tree and extracting a conflict graph, a practical method is proposed to find a valid and initial solution. For solution optimization, an elementary spanning tree transformation based local search algorithm is proposed. To improve the efficiency of optimization, a matrix formulation is also proposed, and by substituting the $\oplus$ for $+$ operations in LP formulation, network simplex method is adopted, where pivoting operations are used to perform optimization. The pivoting properties and two-step successive pivotings are also discussed to further improve algorithm efficiency and optimality.

The experimental results show that on the randomly generated test cases, the matrix-based algorithm shows the highest solution optimality and is five times faster than the elementary transformation method. On the real high-level synthesis benchmarks, the matrix-based method reduces 14% interconnections, while the previous greedy algorithm reduces 8% on average, which implies that interconnection reduction has a large impact on chip performance.

Chapter 4, [**A Multi-Level Algorithm for 3D-IC TSV Assignment in Physical Synthesis**], discusses the problem of Through Silicon Via (TSV) insertion on 3D-ICs in physical synthesis. The problems in this stage are generally large scaled, which lack of efficient algorithms. For example, for large scaled inputs, the execution time of current algorithms may exceed ours or days. Therefore in this topic, the purpose is to propose algorithms with high efficiency, and meanwhile the solution quality is not sacrificed.

Through-silicon via (TSV) assignment problem is one of the key design challenges of 3D-ICs, which is crucial to the wire length and signal delay. In this chapter the TSV assignment problem is formulated as an Integer Minimum Cost Multi Commodity (IMCMC) problem on a IMCMC network. To reduce the huge number of edges of the IMCMC network, a multi-level algorithm is proposed. It first coarsens the IMCMC network level by level, applies a rough flow assignment on each level of coarsened graph, and then generates only promising edges to reduce the IMCMC network size. Benefiting from the multi-level structure, a mixed single and multi commodity flow method is proposed to improve the TSV assignment solution quality. Besides, given a TSV assignment, an extended layer by layer algorithm is proposed to further optimize the TSV assignment.

The experimental results show that the proposed multi-level proposal achieves 37X speedup compared to the previous work, and meanwhile reduces the total wire length by 7.0%, which shows both high optimality and high efficiency. The extended layer by layer optimization further reduces 0.6% total wire length.

Chapter 5, [**Conclusions**], summaries this thesis and addresses some open issues of this research. Some future works are also addressed.

Contents

Listing of figures

Listing of tables

Acknowledgments

I wish to express my sincere appreciation to those who have contributed to this thesis and supported me in one way or the other during this amazing three years.

First of all, I am extremely grateful to my main supervisor, Professor Takeshi Yoshimura. Pursuing a doctoral degree is a hard and long journey, which I obviously cannot finish without the help of my professor. On academic topics, he gives me all his support, showing his guidance and all the useful discussions and suggestions anytime I ask for. His deep insights helped me at various stages of my research. More importantly, he is teaching me how to think, how to learn, and how to be more excellent, and continuously encouraging me to be a better person. In daily life, my professor also gives me as much help as he could, which is the strongest support for my study and life in Japan. I feel so grateful that I could be one of the students of professor Yoshimura, which I believe is the luckiest thing that could ever happen on me in the world.

Secondly, I would like to express my sincere thanks to Professor Watanabe and Professor Kimura, w

.

I also want to express my thanks to all my lab members, Nan Wang, Delong Li, Nan Ding, Hui Zhu, Huanji Sun, Wencan Zhang, Jiayao Wu, Jiayi Ma, and so many friends I did not list here. It is all of you who made my life in Japan gorgeous.

Then, I want to show my deep love to a special association, the Judo circle of the University of Kitakyushu and the Hibikino Judo Club, which opens a totally new world to me and has greatly changed my life. Our couches, Mr. Takenouchi and Mr. Tomohiro, as well as all the members in this club, are always the ones who provide me power, energy and vigour, and are the sources of my fighting will. I love them as teachers, as friends and as families.

Finally, I conserve my deepest thanks to my families. It was their love and encouragement that supported me to finish my doctoral degree and to pursue dreams in the future.

1 Introduction

Very Large Scale integration (VLSI) first appeared in the 1970s, and then quickly began to show evolutional improvements over discrete circuits mainly on cost and performance, and are becoming one of the most indispensable components of the modern technology. Accordingly, designing and developing various functional VLSI, especially by software tools because of the high complexity of modern circuits, are also become of the utmost importance, which is referred to as electronic design automation (EDA), also as electronic computed aided design (ECAD).

Sprouted by the mid-1970s, the EDA industry is going through a rapid growth during the past decades, and is still attracting more and more exploring interests in nowadays. Given the pressing demands for faster and stronger EDA tools especially along with the continuous scaling of semiconductor technology, in this work, main efforts are put in the design field of EDA tools, in order to improve the efficiency and performance of particular methods in the EDA design flow.

In the following parts of this chapter, first, a brief introduction to EDA design flow is given in Section 1.1, as well as some existing problems in current EDA design flow. and in Section 1.2, the motivations and principles of this research are addressed. In Section 1.3, the main contents and common technologies of this research are briefly introduced. Finally in Section 1.6 the organization of this thesis is given.

1.1 Background

1.1.1 EDA Design Flow

EDA design flow is a process which consists of several automated steps to accomplish the design of a VLSI circuit. The general EDA design flow contains several representative steps, as shown in Fig.1.1, including:

1. High-Level Synthesis (HLS) (or behavioural synthesis) — takes architectural design as an input, transforms architectural design descriptions (eg. designs written in C/C++) into register transfer

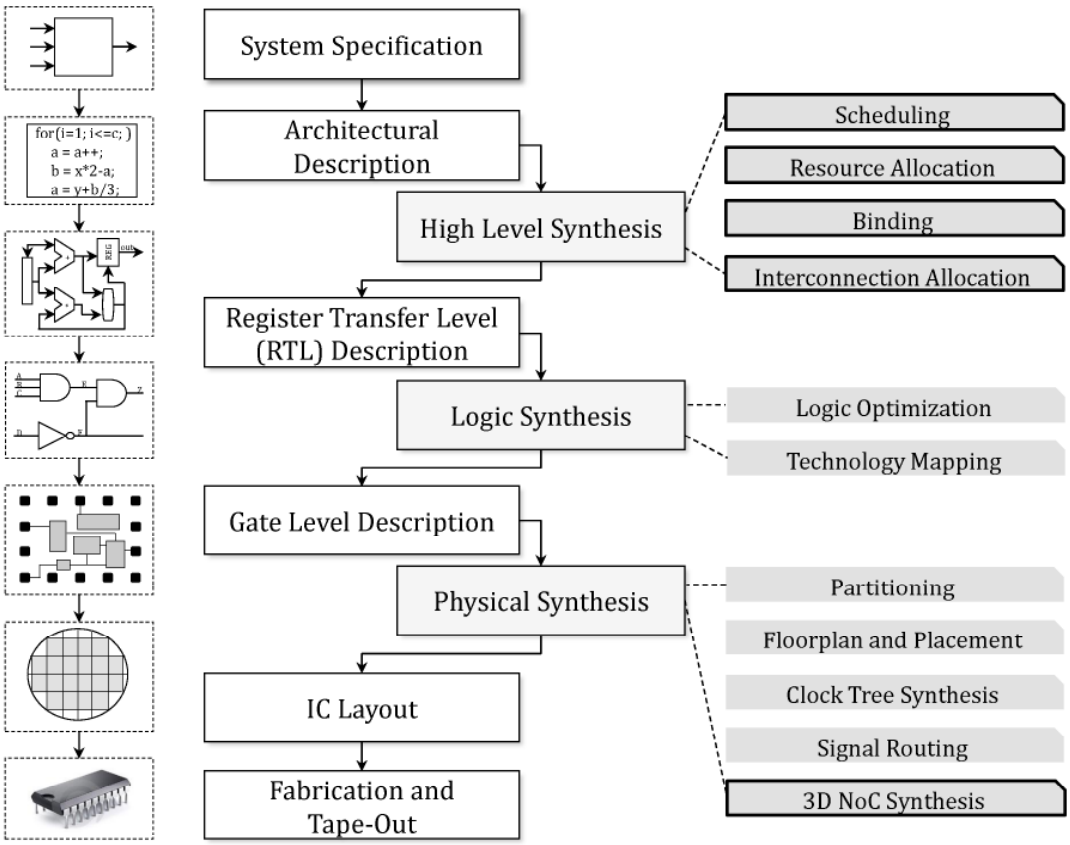


Figure 1.1: A traditional EDA design flow.

level (RTL) descriptions, which is referred to as functional design or logic design;

2. Logic Synthesis — takes the logic design as an input, transforms the RTL description (eg. designs written in Verilog or VHLD) into a netlist of logic gates, which is called circuit design;

3. Physical Synthesis — takes the circuit design as an input, transforms the gate-level circuits into geometric representations with exactly determined locations for circuit components, which are ready for layout;

4. Verification, fabrication and finally Tape-out — last steps in EDA design flow before a chip is manufactured and into market.

During the EDA design flow, up to dozens of design requirements are needed to be considered, such as chip area, power, thermal, routability, signal timing, delay, noise, reliability, etc., which makes it extremely difficult to meet all or a critical part of the requirements while keeping the design cost low and time-to-market fast. Therefore, the development of high efficiency algorithms used in EDA design steps for various optimization objectives are continuously attracting both industrial and academical research interests.

1.1.2 Research Status

The investigations of EDA design methodologies have been carried on for decades since mid-1970s, and significance achievements have been made for automated designs instead of manual designs. Along with the developments of commercial EDA tools, EDA companies are also growing fast and continuously proposing latest design solutions. Following shows a brief review of the current worldwide research situation.

1. **High Level Synthesis:** HLS is one important technology to deal with rapid growing VLSI integration by raising the abstraction level of system descriptions. In recent years, along with the growth of portable and battery-equipped devices, reducing power has became the first-priority design factor. Therefore, the **low power HLS** has became the new hottest topic in HLS.

   Consequently, more and more works are focusing on low power HLS technologies, for example the multiple supply voltage or multiple threshold voltage technology for power reduction, power reduction through interconnection optimization, clock gating and power gating, and dynamic power and frequency scaling, etc. Among them, the proposals for multiple supply or threshold voltage technology, like[10 13 14] and[7 8 23], shows significance achievements in reducing power consumption in the scheduling and binding step of HLS. Besides, works like[43 32 55] try to reduce power by reducing the interconnection complexity, mainly in the interconnection allocation step of HLS.

   Existing issues: Although there are plenty works have been proposed for low power HLS, they are still far from mature. For the researches in low power scheduling and binding, the solution quality is still lower than manual designs, which is expected to be greatly improved; for the researches in interconnection optimization, some important problems such as the port assignment problem, are ignored and not being fully investigated, which is regretful since they do has a great affection on total power consumption.

2. **Logic Synthesis:** Logical synthesis has a much longer history than HLS, which makes its solutions far more mature than HLS. Also, there are a plenty of commercial logic synthesis tools such as the Design Compiler by Synopsys, Encounter RTL Compiler by Cadence Design Systems, HDL Designer by Mentor Graphics, Quartus II integrated Synthesis by Altera, XST (delivered within ISE) by Xilinx, etc. There are also open source tools like the Logic Synthesis Tool (SIS)[47]. In academic researches, there are several major topics including combinational and sequential logic synthesis, synthesis for asynchronous circuits, synthesis for elastic circuits, automatic pipelining, etc[46].

   Existing issues: As pointed out in[46], equivalence checking problem still exists in the transformations that modify timing beyond the natural boundaries of RTL state signals. Besides, the automation for out-of-order execution is also pointed out as another challenge.

3. **Physical Synthesis:** Similar to logic synthesis, physical synthesis also has a long history with several mature commercial tools. such as the RTL Compiler/Build Gates/Physically Knowledgeable Synthesis (PKS) by Cadence, and the Design Compiler by Synopsys. Therefore, the solution quality is satisfied for traditional synthesis, such as partitioning,

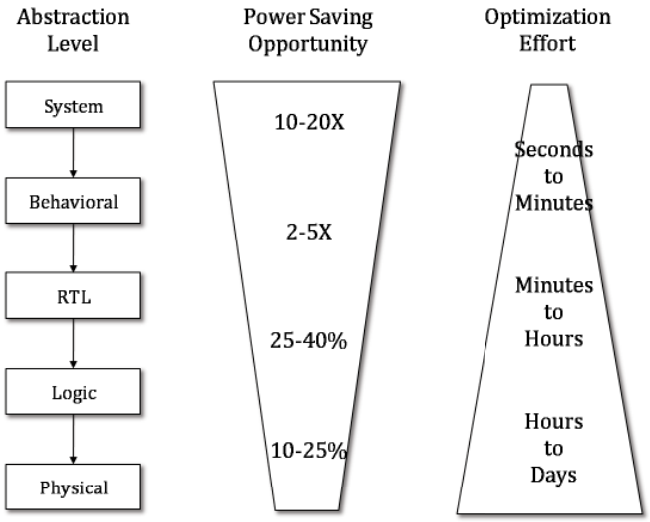


Figure 1.2: Power saving opportunity and effort at different levels of abstraction from[5]

floorplan and placement, clock tree synthesis and routing. On the other hand, along with the development of 3D ICs, the physical synthesis for 3D IC, or 3D Network-on-Chip (NoC), has became a new topic in the physical synthesis.

In the physical synthesis for 3D IC, one important topic is the Through Silicon Via (TSV) insertion (also called the TSV assignment problem), because the TSVs are very large compared to logic gates, say dozens to hundreds of times the area of a standard cell, and the TSV assignment is crucial to the wire length and signal delays of 3-D circuits. In the related works of TSV assignment, some works handle TSV insertion as an independent problem, whose optimization objectives include: total wire length, chip thermal, interconnection complexity, etc. Some other works perform TSV insertion together with floorplan or placement, where TSVs are pre-planned during floorplan or placement, and are once more optimized after the floorplan or placement is finished. These works imply that a careful TSV assignment can greatly reduce total wire length, chip area or improve thermal issues.

Existing issues: One of the problems in TSV assignment is, together with the growing design and circuit size, the TSV assignment problem size is also becoming huge, which makes it difficult to be solved using existing algorithms. For example in the work[76], since the problem formulation is extremely huge, the authors proposed several methods to reduce the formulation size by scarifying the solution optimality. Another work[77] proposed a Lagrangian based algorithm but it cannot be applied on large testcases.

1.2 Research Motivations and Principles

Motivated by the existing issues in EDA design flow, in this research, several problems in two parts are to be studied. One is the **low power HLS** technology, including the scheduling, resource allocation and binding, and interconnection optimization. Another is the **physical synthesis for 3D IC** technology, w

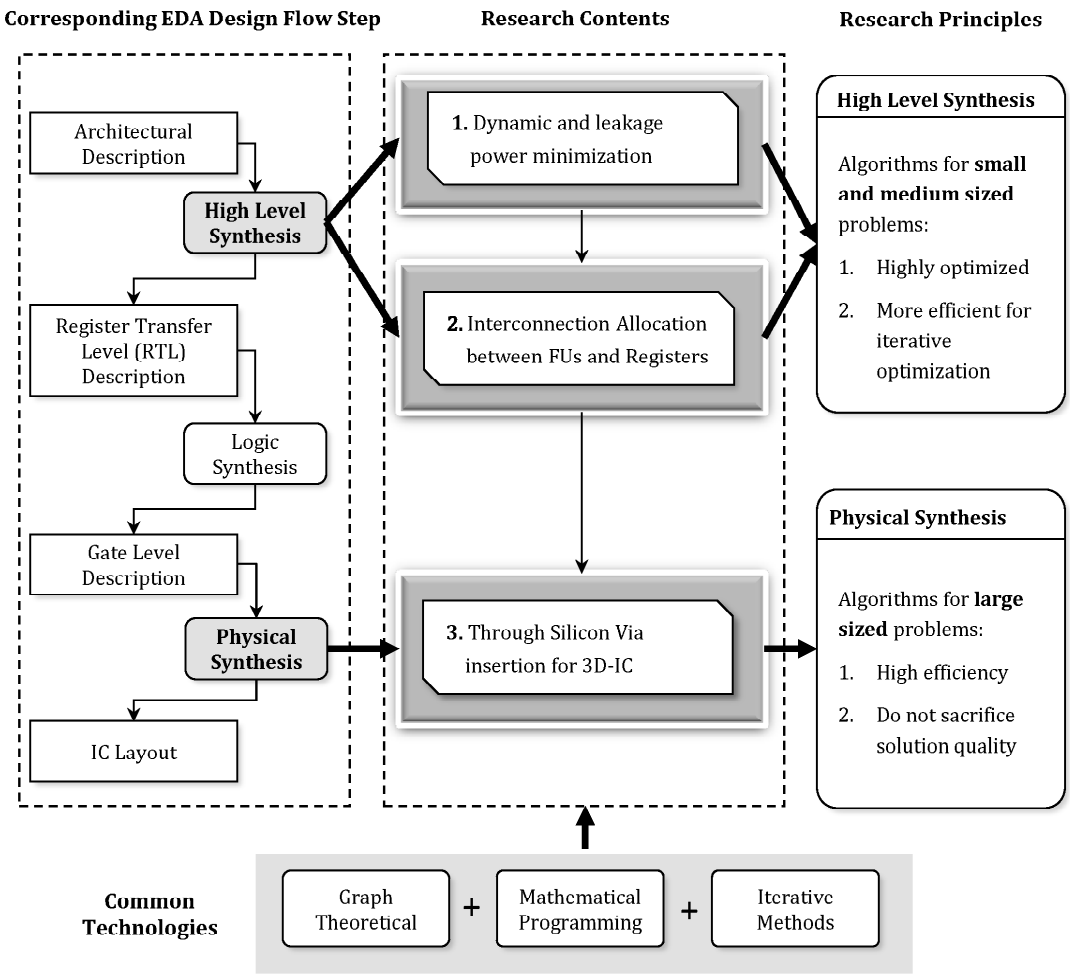


Figure 1.3: research contents

1. Both low power HLS and physical synthesis for 3D IC technologies are of great importance with worth deep investigation. As pointed out in[5], attempting to reduce power at RT level typically requires much more efforts than at behaviourial or system level, and as shown in Fig.1.2, the higher abstraction level is, the less optimization effort is needed, and the higher power saving opportunity is. Therefore exploration for low power HLS is a promising topic both now and in the future. On the other hand, physical synthesis for 3D IC is an indispensable step for impelling real applications of 3D IC chips in industry, which is a promising solution for higher and higher IC integration.

2. There are open issues for low power HLS and physical synthesis for 3D IC to be solved, as pointed out in Section 1.1.2. In HLS, it is still difficult for the automated design algorithms to produce optimal solutions better than manual designs for even small or medium sized HLS problems. Therefore, **more optimized algorithms in HLS are required.** In physical synthesis, along with the design size growing, it is difficult for the current existing algorithms to handle huge sized problems. Therefore, **more efficient algorithms for huge sized problems are required.**

   Essentially, these open issues for low power HLS and physical synthesis for 3D IC are essentially the same, which is to propose **high efficiency** algorithms. In order to improve the algorithm optimality, one important and effective way is to use iterative optimization: assume the possibility to get optimum solutions in one iteration is $p(0 \leq p \leq 1)$, if the algorithm is executed for n iterations with random restart, the possibility to get optimum solutions is $p^* = 1 - (1 - p)^n$. Obviously the larger n is, the closer p^* is to 1. This requires in each iteration, the algorithm must be efficient enough to produce solutions fast. Therefore, fast and highly optimized algorithms are still expected.

3. Moreover, the problems for both low power HLS and physical synthesis for 3D IC share similar formulations, which allows similar methodologies to be adopted. To be specific, all these problems can be formulated by Integer Linear Programming (ILP), one of the methods in **mathematical programming**, and written into $A \cdot x = B$ where x are integers. In these problems, matrix A happens to be the incidence matrix of a graph. Therefore, if the constraints that x are integers can be relaxed (for example if A is totally unimodular), the ILP formulations can be transformed into Linear Programming (LP), and therefore **graph theories** can be considered. Besides, since one of the purposes is to improve optimality, **iterative methods** are also expected. Consequently, **common technologies** can be applied to these problems, to improve the efficiency and optimality of their solutions.

1.2.1 Research Principle

Therefore, the principle of this research is stated as the following:

By utilizing **graph theory** and **mathematical programming** methods, which are widely used methodologies in EDA design area,

1. **For small and medium sized problems in HLS**, say the scheduling and binding problems, algorithms that produce solutions as optimal as possible (optimal or sub-optimal), which excel manual designs, are to be proposed. For this purpose, since iterative optimizations are expected to improve solution quality, algorithms also expected to be efficient for one iteration.

2. **For huge sized problems in physical synthesis of 3D-IC** which are impossible for manual design or with no existing practical solutions, for example in floorplan and placement step in physical synthesis, algorithms which are efficient enough to produce solutions in reasonable time with no or slight optimality degrade are to be proposed.

1.3 Research Contents and Common Technologies

In this section, the principles of this research, as well as main contents, common key technologies and contributions are introduced. Fig.1.3 gives an overview of the organization of this research. There are three parts in this research, corresponding to particular design steps in EDA design flow. The first two parts are the works in HLS, and the third part is the work in physical synthesis. The three works share common key technologies including **graph theory**, **mathematical programming** and **iterative optimization**, as shown in the right part of Fig.1.3.

In the following, three works are briefly introduced respectively, as well as their common and specified core technologies shown in Fig.1.4.

1.3.1 Dynamic and leakage power minimization[3]

The dynamic and leakage power minimization under multi-V_{dd} or/and multi-V_{th} corresponds to the highest step, operation scheduling in HLS procedure. It solves the problem that, how to determine the schedule and the supply/threshold voltage for each operation, to reduce the overall dynamic/leakage power consumed by functional units. The goal is to propose a new algorithm, which can excel previous works in both efficiency and solution optimality. Besides, the combined scheduling and binding for power minimization is also discussed.

Technologies: it is first given mathematical formulation, the integer linear programming (ILP), and then is relaxed to linear programming (LP) formulation. Although general linear programming methods can solve this problem, given the special properties of its constraint graph, the network simplex method, one of the graph theoretical methods, is adopted to improve the algorithm efficiency. Further, to improve solution quality, an iterative method, local search based optimization with random restart, is adopted.

1.3.2 Interconnection allocation between FUs and registers[35]

The research of this part corresponds to the interconnection allocation step in HLS procedure. It solves the problem that, after functional unit and register binding, how to determine the interconnections

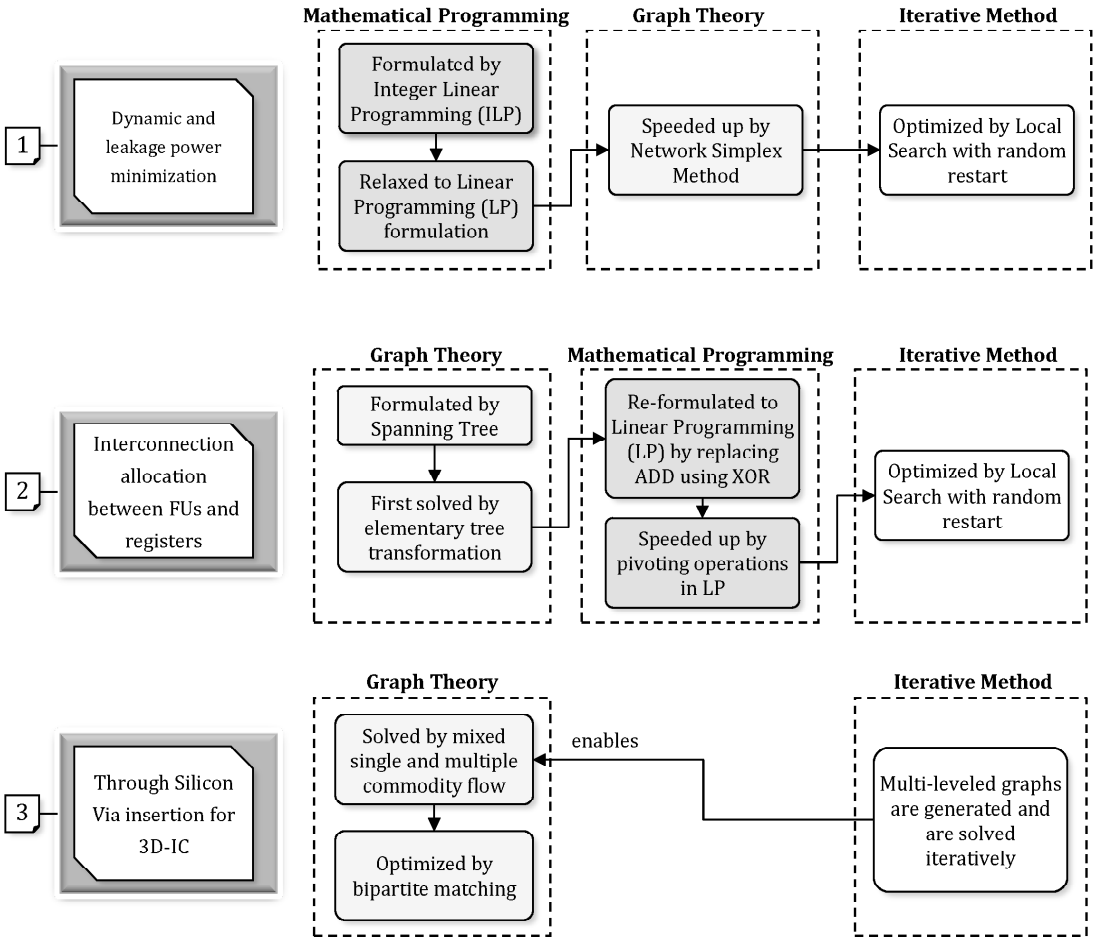


Figure 1.4: Common Technologies: graph theory, mathematical programming and iterative methods

between registers and functional unit ports, in order to minimize the interconnection complexity and multiplexer size, and further to reduce the interconnection power and chip area.

Technologies: it is first formulated as a graph with spanning tree, and solved using graph theoretical methods including elementary tree transformation and minimum vertex cover. Then, to improve algorithm efficiency, the problem is reformulated as LP problem by replacing adding operations in standard LP by exclusive-or operations. By doing so, the algorithm is speeded up by pivoting operations in standard LP, which is one of mathematical programming methods. Finally, to improve solution quality, similar to work 1, an iterative method, local search based optimization with random restart, is also adopted.

1.3.3 Through silicon via (TSV) insertion for 3D-IC[71]

TSV insertion for 3D-IC is the physical synthesis for 3D ICs. It solves the problem that, given a floorplan of modular on a 3D chip and a net list, how to insert TSVs between chip dies, such that the total wire length that connecting signal pins and TSVs is minimized. More importantly, when the problem size is huge, i.e., over one thousand nets and more than 3 chip dies, how to solve the problem

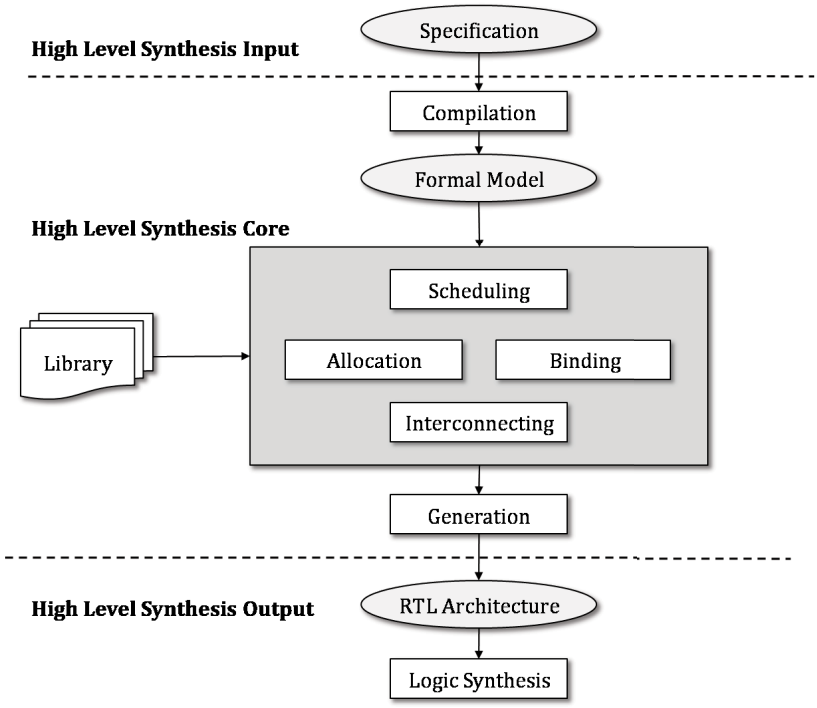


Figure 1.5: Basic concept of High Level Synthesis.

within reasonable execution time without sacrificing the solution quality, while the current existing algorithms are not able to handle.

Technologies: it is first formulated by a multi-level graph, and is solved by mixed single and multiple commodity min-cost flow. To improve solution quality, the bipartite matching is adopted, which is also a method of graph theory.

1.4 Preliminaries of High Level Synthesis

High-level synthesis (HLS), also referred to as behavioral synthesis, is an automated design process that interprets an algorithmic description of a desired behavior and creates digital hardware that implements that behavior[2]. Fig.1.5 shows the basic concept of HLS. The inputs of HLS are specifications described by languages with system-level abstractions such as ANSI C, SystemC, C++ or SystemVerilog. During synthesis process, the input codes are first compiled and analyzed, and procedures including operation scheduling, resource allocation and binding are went through. Then a register-transfer level (RTL) hardware description is generated, which is the outputs of the HLS, containing data path components including registers, multiplexers, functional units, buses, and controlling logics. Finally logic synthesis is conducted on the RTL structure.

1.4.1 Necessity of High Level Synthesis

HLS technology begins to attract research interests since the late 20th century, and now is still one of the hottest topics in current EDA field. One of the reasons is, it enables IC designers to describe the design at a higher electronic system-level (ESL), and to efficiently build and verify hardware even

without specific hardware knowledge.

Another reason for HLS being strongly required is, the ESL paradigm is shifting to higher level of abstractions caused by the rise of system complexities. To be more exact, along with the integration of ICs growing rapidly in recent decades, the vary-large-scale integrated circuits (VLSI) and the ultra-large-scale integration circuits (ULSI), which have more than 1 million transistors, are being proposed and begin to put into production. The huge size of target circuits makes it impossible for manual design, especially from lower circuit levels due to the extremely huge number of circuit components. As a result, designing from higher circuit levels is an inevitable choice, that is, *fast IC scaling trend strongly calls for HLS*.

More importantly, in recent years, HLS is not only necessary for large scale designs; moreover, it is especially required for low power IC designs because of the outstanding performances that HLS technology has shown in a series of low power works, which is referred as to low power high level synthesis.

1.4.2 Necessity of Low Power High Level Synthesis (HLS)

Along with the rapid growth of portable and battery-equipped devices, reducing power consumption has became the primary limiter of scaling semiconductor process technologies and adding features to integrated circuits. As a result, the IC design is undergoing a significant transition from performance oriented to power oriented, where power has became one of the first-priority design factors[5]. This transition invokes many researches working on low power techniques on register transfer level, such as power gating, clock gating, multiple supply or threshold voltage techniques, and recently approximate computing resources in error-tolerant applications.

However, as pointed out in[5], attempting to reduce power at RT level typically requires much more efforts than at behaviourial or system level, because making decisions at higher levels usually result in larger impact on final designs. As shown in Fig.1.2, the higher abstraction level is, the less optimization effort is needed, and the higher power saving opportunity is. Consequently, *reducing power from higher level imperatively calls for HLS techniques*.

1.4.3 Related Low Power HLS Works

Major Power Dissipations

There are broadly three kinds of power dissipations of a circuit: switching power (also referred to as dynamic power), static power (also called leakage power) and short-circuit power, Among the three, dynamic power, denoted as P_{dy}, and leakage power, denoted as P_{lk}, are much higher than the short-circuit power, and attract most research interests.

- Dynamic power is consumed through circuit activities, e.g. when switching takes place on logic gates, computed as:

$$P_{dy} = C_L \times V_{dd}^2 \times f \times sw \tag{1.1}$$

where C_L is the output capacitance, V_{dd} is the supply voltage, f is the clock frequency, and sw is the switching activity. Easy to see, P_{dy} is positively related to the supply voltage V_{dd} of logic gates, f and sw. Therefore, one effective way in HLS procedure to reduce dynamic power is to use multiple supply voltage (multi-V_{dd}) technology.

- Leakage power is primarily the result of unwanted subthreshold current in the transistor channels, and is consumed whenever circuit components are busy or idle. It has became more dominating than dynamic power especially in in deep sub-micron process technology (65nm and below), say more than 50% of the total IC power consumption[50], which makes it a top concern for current IC design. The subthreshold-driven leakage power is strongly influenced by variations in the transistor threshold voltage, denoted as V_{th}. Therefore, lowering threshold voltage, or using dual or multiple threshold voltage (multi-V_{th}), is effective to reduce leakage power.

Previous Low Power Technologies

There has already been a large amount of investigations on low power HLS techniques, to reduce either dynamic power or leakage power, which focus on one or several particular steps in HLS, such as operation scheduling, resource allocation or interconnection allocation. In the following several typical categories of low power technologies as well as their representative literatures are listed.

- **Multiple supply voltage (multi-V_{dd})**. The multiple supply voltage technology is mostly applied on operation scheduling or binding in HLS procedure, like[10 11 12 13 14]. It shows promising results in reducing dynamic power by assigning different V_{dd} to operators. Chen et al.[10] presented a network flow based dual-V_{dd} binding algorithm for low-power resource binding with switching activity being considered. Liu et al.[12] addressed a voltage partitioning problem arising in multiple supply voltage design during HLS. They proved it as NP-hard, and proposed an efficient approximation algorithm with linear time-complexity for practical designs, which is tens to hundreds of times faster than previous works. Jiang et al.[13] discussed a multiple supply voltage scheduling problem in HLS to minimize the system power consumption. They formulated it as a linearly constrained separable convex optimization problem, and solved it using the integer time budgeting technique, which produced near-optimal results.

- **Multiple threshold voltage (multi-V_{th})**. The multiple threshold voltage technology is also applied on scheduling or binding like[7 8 9 23], where operations are first scheduled and are assigned to different types of functional units (FU) executed under different threshold voltages. In[7] a greedy method was proposed that prioritized modules with largest leakage reduction potentials for high-V_{th} implementation. In[8] a composite constraint graph was proposed to present resource binding constraints, together with a maximum weight independent set based scheduling algorithm. In[9] leakage power was minimized by characterizing the bounded delay degradations for FUs with multi-V_{th} values. The latest work[23] proposed a binding conflict

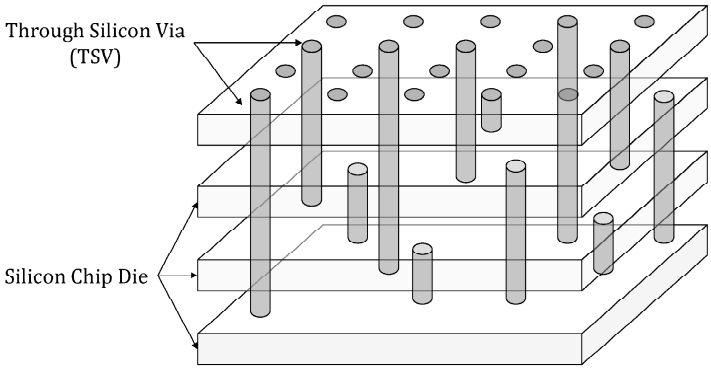


Figure 1.6: An example of 3D-IC and Through Silicon Vias (TSV).

graph based scheduling method to minimize the number of FUs. They all showed that multi-V_{th} approach in HLS can significantly reduce leakage power dissipation.

- **Power Reduction through Interconnection Optimization**. Since the interconnections also consume a large fraction of leakage power, minimizing the interconnection complexity also reduces power consumption. For example in work[39], the interconnections between functional units and registers are discussed. In other works[40] and[41], other interconnections, for example multiplexers, are also considered.

- Other technologies like clock gating[42], power gating[44 45], d

.

1.5 Preliminaries of 3D-IC and Through Silicon Via Insertion

Three-dimensional integrated circuit (3-D IC) is a promising IC manufactural technology with stacks of dies that approaches higher density, reduced power, smaller footprint, improved performance and lower cost compared with traditional monolithic ICs, and many important works on 3-D IC technology are proposed in recent years[65 66]. The 3-D IC technology uses through-silicon vias (TSV) to provide vertical electrical connections passing through a silicon wafer or die. The TSV is an emerging interconnection technology that will replace the traditional wire-bonding process in chip/wafer stacking, to increase inter-die communication bandwidth, reduce form factor, and lower power consumption of stacked multi-die systems by eliminating the need of long cross-chip interconnects existing in 2-D ICs. Fig.1.6 shows an example of 3D IC die stacks with TSVs.

One of the key design challenges of 3-D IC is the optimization of the number and locations of TSVs, which is generally called the TSV assignment problem. Given a 3-D IC netlist describing the inter-die nets, TSV assignment is to decide which TSVs are used to implement the nets spanning different chip dies. After the TSV assignment, routing is applied on each die to complete the electrical connection of every net. Since the TSVs are very large compared to logic gates, say dozens to hundreds

of times the area of a standard cell, the TSV assignment is crucial to the wire length and signal delays of 3-D circuits, and it is now attracting broad interest among both academic and industrial researchers.

1.6 Thesis Organization

This thesis consists five chapters. The rest of this thesis is organized as follows.

In chapter 2, [**A Unified Scheduling Approach with Multiple V_{dd} or/and V_{th} in HLS**], the dynamic and leakage power minimization problem using multiple V_{dd} and V_{th} technology in operation scheduling is discussed. The combined scheduling and binding method is also discussed in this chapter.

In chapter 3, [**Interconnection Allocation Between Functional Units and Registers in HLS**], the interconnection optimization techniques in HLS, which has not been fully investigated, is discussed in this chapter, and several algorithms are proposed for the port assignment problem between functional units and registers.

In chapter 4, [**A Multi-Level Algorithm for 3D-IC TSV Assignment in Physical Synthesis**], the problem of Through Silicon Via (TSV) insertion on 3D-ICs in physical synthesis is discussed, which has no practical algorithms when the problem size becomes huge. In this chapter the methodologies and graph theories used in HLS are utilized in physical synthesis stage.

In chapter 5, [**Conclusions and Future Work**], this thesis is concluded and some future works are given.

2

A Unified Scheduling Approach with Multiple V_{dd} or/and V_{th} in HLS

In this chapter, the first topic, dynamic and leakage power minimization, is to be discussed, which corresponds to the scheduling and binding step in HLS. It determines the supply voltage or threshold voltage for operations and FUs, as well as the scheduling for operations, to minimize the total power consumption. The purpose is to propose algorithms to get optimum or near-optimum solutions. The problems are first formulated by Integer Linear Programming, and by relaxing to Linear Programming, the network simplex method is adopted iteratively to solve the LP problem for optimization. The detailed formulations and algorithms are discussed in the following of this chapter.

2.1 Background and Contributions

Low power integrated circuit (IC) design, in the recent ten years, has been a substantial research theme because of the fast growth of portable computing devices and reliability requirements under higher operation temperature. The major power dissipation sources include leakage power, short circuit power and dynamic power, in which the leakage power and dynamic power are more dominating than short circuit power. The threshold driven leakage power is strongly influenced by the transistor threshold voltage V_{th}, and the dynamic power decreases with the decreasing of supply voltage V_{dd}. Consequently, it is promising and popular to use dual or multiple threshold voltage and/or multiple supply voltage technique for power reduction, by trading off the performance of less timing-critical logics[5]. For example, there are many approaches that minimize both the leakage and dynamic power in generic digital CMOS designs[15][16][17][18] by determining the V_{th} and V_{dd} of each module or logic gate in gate level.

Except for the CMOS level optimizations under multiple-voltage technologies, there are also many power scaling efforts have been made in the high-level synthesis (HLS) stage, because it was pointed out that power saving opportunities at behavioral and system levels are much higher than that at register-

transfer level (RTL)[6]. One of the most powerful techniques is multiple voltage scheduling, which allows functional units (FU) work with different threshold voltages V_{th}, or different supply voltages V_{dd}, with different execution delays.

The multiple supply voltage scheduling approaches[10 11 12 13 14] showed promising results in reducing dynamic power by assigning different V_{dd}. Chen et al.[10] presented a network flow based dual-V_{dd} binding algorithm for low-power resource binding with switching activity being considered. Liu et al.[12] addressed a voltage partitioning problem arising in multiple supply voltage design during HLS. They proved it as NP-hard, and proposed an efficient approximation algorithm with linear time-complexity for practical designs, which is tens to hundreds of times faster than previous works. Jiang et al.[13] discussed a multiple supply voltage scheduling problem in HLS to minimize the system power consumption. They formulated it as a linearly constrained separable convex optimization problem, and solved it using the integer time budgeting technique, which produced near-optimal results.

The multiple threshold voltage scheduling approaches[7 8 9 23] schedule the operations and assign different threshold V_{th} voltages. In[7] a greedy method was proposed that prioritized modules with largest leakage reduction potentials for high-V_{th} implementation. In[8] a composite constraint graph was proposed to present resource binding constraints, together with a maximum weight independent set based scheduling algorithm. In[9] leakage power was minimized by characterizing the bounded delay degradations for FUs with multi-V_{th} values. The latest work[23] proposed a binding conflict graph based scheduling method to minimize the number of FUs. They all showed that multi-V_{th} approach in HLS can significantly reduce leakage power dissipation.

Although a great quantity of literatures have been proposed for multiple voltage problems, further studies for multi-V_{dd} and multi-V_{th} scheduling are still needed. For example for multi-V_{dd} problem,[12] fully studied voltage partitioning but not operation scheduling;[13] solved the scheduling problem but did not consider the resource usage. For multi-V_{th} problem,[9] targeted in resource binding rather than scheduling, and the efficiency of the scheduling algorithm in[23] is a big concern. In addition, in previous works for leakage power reduction, either scheduling or binding is considered independently, but there is no work that considers two steps jointly. Accordingly, in this work, the multi-V_{dd} problem and the multi-V_{th} problem are first solved separately, and then discuss the impact of binding on the overall power including the interconnection power. Besides, for multi-V_{dd} and multi-V_{th} problems, a unified approach is proposed, which provides an opportunity that the dynamic and leakage power can be minimized simultaneously, which is believed to be worth studying.

In this paper, my previous work[4] is extended to propose a unified scheduling scheme that works both for leakage and dynamic power optimization, given multiple threshold or/and multiple supply voltages. In[4], a dynamic power and resource co-optimization problem is solved under multiple supply voltage using Network Simplex Method; the additional contributions in this work are stated as:

- **A unified scheduling scheme** is proposed for three different optimization problems: (1) dynamic power and resource co-optimization; (2) leakage power optimization; (3) dynamic power and leakage power co-optimization.

- It is revealed that the three problems with multiple objectives share two common sub-problems: operation delay assignment and resource density variance minimization; the two sub-problems are simultaneously solved using **vertex potentials** proposed in this work.
- For leakage power optimization, **a post processing** is proposed to eliminate the limitation of the proposed scheduling algorithm when being utilized for leakage power.
- **Functional unit and register binding** is combined into operation scheduling, to reduce multiplexer and register power, thus to reduce overall power consumption.
- It is shown in the experiments that, when the proposed unified scheduling scheme is utilized for leakage power minimization, it significantly excels the latest existing work both in terms of power and execution time.

The rest of this chapter is organized as follows. Section 2.2 gives the problem formulation, Section 2.3 motivates this work, Section 2.4 solves the delay assignment problem, and Section 2.5 solves the resource density minimization problem. Section 2.7 introduces the unified scheduling scheme, and Section 2.8 utilizes it on three problems. Section 3.6 shows the experimental results, followed by conclusions in Section 2.10.

2.2 Problem Formulations

In this section, the inputs, outputs, and objectives of three problems are clarified, followed by a unified Integrated Linear Programming (ILP) formulation for three problems.

2.2.1 Problem Description

According to[23], for a FU under a fixed supply voltage V_{dd}, its leakage power decreases, and delay increases monotonously along with the increasing of threshold voltage V_{th}; according to[13], for a FU under a fixed V_{th}, its dynamic power decreases, and delay increases monotonously along with the decreasing of V_{dd}. In this work, the schedulings under a fixed V_{th} and multi-V_{dd}, and under a fixed V_{dd} and multi-V_{th}, are mainly discussed. Since the delay-voltage curve of a FU is monotonous, to unify V_{dd} and V_{th}, the FU delays are considered instead of their real voltage values during scheduling; after scheduling, the real voltages for FUs can be known according to their delays.

The **inputs** of the scheduling algorithm include:

- A data flow graph (DFG) $G = (V, E)$ of a behavioral description, where V is the set of operations and E is the data dependencies. Each operation $op \in V$ has a type $\tau_{op} \in \Gamma$, where Γ represents all operation types such as addition, multiplication, etc.
- A discrete function describing the *leakage* power of FUs under fixed V_{dd} and different V_{th}, denoted as $LK_\tau(d), \tau \in \Gamma$. d is the FU delay under a certain V_{th}, given in the number of clock cycles, i.e., the number of control steps, where $d_{min} \leq d \leq d_{max}$. $LK_\tau(d)$ is the leakage power of a FU of type τ and delay d.

Table 2.1: Dynamic Power-Delay Table for ADD and MUL under clock frequency of 769MHz (clock period 1.3ns)

	$P_{dynamic}$ (μW)	Exec. Time	V_{DD} (volts)	Delay (clock cycles)
ADD	195.46	1.2ns	1.1	1
	62.44	2.0ns	0.9	2
	34.67	2.7ns	0.7	3
MUL	2259	2.3ns	1.1	2
	723	3.9ns	0.9	3
	384	5.2ns	0.7	4

T .2: Leakage Power-Delay Table for ADD and MUL under clock frequency of 769M (clock period 1.3ns)

	P_{leak} (μW)	Exec. Time	V_{th} (volts)	Delay (clock cycles)
ADD	160.05	1.1ns	0.18 (low-V_{th})	1
	7.11	1.4ns	0.32 (high-V_{th})	2
MUL	270.71	2.1ns	0.18 (low-V_{th})	2
	27.40	3.4ns	0.32 (high-V_{th})	3

- A discrete function describing the *dynamic* power of FUs under fixed V_{th} and different V_{dd}, denoted as $DY\ (d), \tau \in \Gamma$. d is the FU delay under a certain V_{dd}, where $d_{min} \leq d \leq d_{max}$. $DY\ (d)$ refers to the dynamic power of a FU for an *active execution*.

The **constraint** is the scheduling latency, denoted as T_{con}, which is the maximum allowable number of control steps to finish the execution of data flow.

The **outputs** include: (1) a delay assignment for each $op \leftarrow d, op \in V$ and (2) a scheduling for each $op \leftarrow [1, T_{con}], op \in V$ such that all the dependency constraints are satisfied.

The **objectives** for three problems are separatively stated as:

- Problem 1. dynamic power and resource co-optimization: to minimize total dynamic power of all operations, then reduce the number of FUs;
- Problem 2. leakage power optimization: to minimize total leakage power consumption of all FUs;
- Problem 3. dynamic and leakage power co-optimization: to minimize dynamic power as well as leakage power.

The leakage and dynamic power and delay data used in this work are shown in Table 4.1 from[25] and Table 2.1 from[26]. All data are obtained by HSpice simulation in 90nm TSMC CMOS technology, under the clock frequency of 769MHz (clock period 1.3ns), 16-bit data width. The delays of FUs are given in numbers of control steps which are integers. In additional, to adopt linear programming, which is discussed in Section 2.4.1, the integer delays for FUs shall be consecutive. For example in Table 2.1 the possible delays of an adder are $\{1, 2, 3\}$, and of a multiplier are $\{2, 3, 4\}$.

2.2.2 Unified ILP Formulation

Given a DFG $G = (V, E)$, for each operation $op_i \in V$, two variables p_i and q_i are defined, which means op_i starts at time p_i and ends at time q_i. Here the time means the time point between two control steps, as the example shown in Fig.2.1(b). The constraints of ILP formulation can be expressed as:

$$\forall e = (op_i, op_j) \in E: \quad q_i - p_j \leq 0 \tag{2.1}$$

$$\forall e = (s, op_i) \in E: \quad s - p_i \leq 0 \tag{2.2}$$

$$\forall e = (op_i, t) \in E: \quad q_i - t \leq 0 \tag{2.3}$$

$$\forall op_i \in V: \quad d_{min}^{op_i} \leq q_i - p_i \leq d_{max}^{op_i} \tag{2.4}$$

$$0 < t - s \leq T_{con} \tag{2.5}$$

where $s, t, p_i, q_i, i = 1 \cdots n$, are integers, and variables s and t indicate the beginning and ending time of the execution of data flow. Eq.2.1 to Eq.2.3 are data dependency constraints, Eq.2.4 are delay constraints, and Eq.2.5 is latency constraint.

To express the objectives of three problems, denote the total dynamic power as TP_{dy}, the total leakage power as TP_{lk}, and the resource usage cost as RES, defined in Eq.2.7 to Eq.2.9.

For dynamic power TP_{dy}, it is calculated as the summation of dynamic powers of all operations, since dynamic power is consumed only when a FU is active. For leakage power TP_{lk}, it is estimated using the numbers of FUs after scheduling and FU allocation, since leakage power is consumed whether a FU is idle or active.* The number of FUs of type τ and delay d, denoted as $N(d, \tau)$, is the maximum $N_t(d, \tau)$ in all control steps:

$$N(d, \tau) = max\{N_t(d, \tau)\}, 1 \leq t \leq T_{con} \tag{2.6}$$

where $N_t(d, \tau)$ is the number of active operations of type τ and delay d at control step t.

Besides, the overall leakage power consumption, including multiplexers and registers, is also to be evaluated by combining binding into scheduling, which is discussed in Section 2.7.3.

R

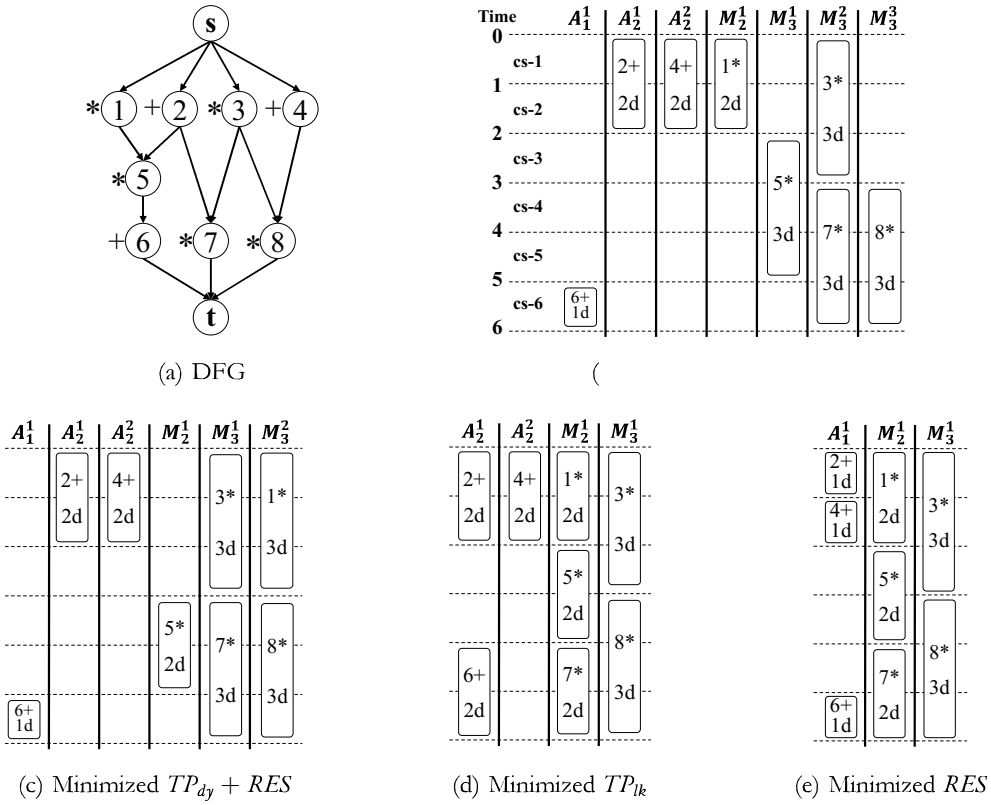
. For example, RES is the total number of FUs if β_d are all set as 1, or is the approximate total area of all FUs if β_d are set as the area value of FU of type τ and delay d.

Therefore, the dynamic power TP_{dy}, leakage power TP_{lk} and resource usage RES are calculated as:

$$TP_{dy} = \sum_{op_i \in V} DY \quad _{op_i}(d(op_i)) \tag{2.7}$$

$$TP_{lk} = \sum_{\in \Gamma} \sum_{d_{min} \leq d \leq d_{max}} LK \quad (d) \cdot N(d, \tau) \tag{2.8}$$

*Don't consider the power gating technology in this work.



(a) DFG (

(c) Minimized $TP_{dy} + RES$

(d) Minimized TP_{lk}

(e) Minimized RES

Figure 2.1: Different scheduling and delay assignment results for: (b) dynamic power minimization, (c) dynamic power and resource usage co-optimization, (d) leakage power minimization, and (e) resource minimization only. In the figures $1d$ means the delay of operation is 1, etc; cs-1 means control step 1, etc.

$$RES = \sum_{\tau \in \Gamma} \sum_{d_{min} \le d \le d_{max}} \beta_d \cdot N(d, \tau) \tag{2.9}$$

where $d(op_i)$ is the delay of op_i, DY is the dynamic power function, LK is the leakage power function, β_d is the weight for FU of type τ and delay d.

The objectives of three problems are written as:

- problem 1: $min \rightarrow TP_{dy} + RES$
- problem 2: $min \rightarrow TP_{lk}$
- problem 3: $min \rightarrow TP_{dy} + TP_{lk}$

2.3 Preliminaries and Motivations

In this section it is first explained that the three problems share two common sub-problems, **operation delay assignment** and **resource density variance minimization**; then some preliminaries including **mobility graph**, **mobility allocation** and **vertex potential** are introduced and motivated.

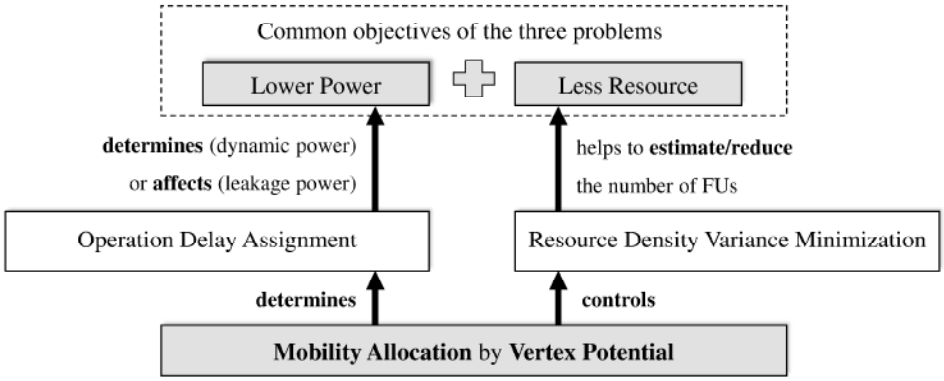


Figure 2.2: Three problems share two common sub-problems: delay assignment and resource density variance minimization. Mobility allocation by vertex potential is proposed to optimize the two problems simultaneously.

2.3.1 Two Common Sub-problems

Given three problems, Fig.2.1 shows an example to illustrate the differences and similarities of their optimization objectives.

First, Fig.2.1(a) shows a DFG; Fig.2.1(b) shows the minimized dynamic power, where as large as possible delays are assigned to operations within latency constraint. Fig.2.1(c) shows the dynamic power and resource usage co-optimization (problem 1), where the dynamic power is the same as Fig.2.1(b) but the number of FUs reduces by 1. Fig.2.1(d) shows the minimized leakage power (problem 2), where the number of FUs is smaller than that in Fig.2.1(c) because leakage power is primarily determined by the number of FUs; meanwhile, operators with larger delays are preferable, for example two 2-delay adders ($\mathcal{A}_2^1$ and $\mathcal{A}_2^2$ in Fig.2.1(d)) consume less power than one 1-delay adder ($\mathcal{A}_1^1$ in Fig.2.1(e)). Problem 3 can be regarded as a combination of problem 1 and 2 with conditions.

Consequently, as illustrated in Fig.2.2, three problems share two common objectives: lower power and less resource; they are respectively realized by two common sub-problems:

Operation Delay Assignment

dynamic power is determined by, and leakage power is greatly affected by the delays of operations, because intuitively FUs with larger delays consume less power, as shown in Table 4.1 and 2.1; therefore it is preferable that *as large as possible delays can be assigned to each operation.*

R

.

G

.5.1. As known from Eq.2.6, in

order to minimize $N(d, t)$, the number of FUs, it is preferable to *uniformly distribute resource density among all control steps*, i.e., to minimize the resource density variance.

Consequently, the common objective for three problems is:

$$min: \ P + \gamma \cdot R \tag{2.10}$$

where P is the objective function of the first sub-problem, operation delay assignment, and R is the objective function of the second sub-problem, resource density variance minimization. γ is the trade-off parameter between power and resource which is discussed in the following sections. To solve the two sub-problems P and R simultaneously, the **mobility allocation** and **vertex potential** are proposed in this paper. I first show the motivations for mobility allocation and vertex potential, and give their definitions in Section 2.4 and Section 2.5, respectively.

2.3.2 Mobility and Mobility Overlap

In HLS scheduling problems, the *mobility of operation* are proposed mainly for operation probability estimation[20]:

DEFINITION 1 The *mobility* of operation op_i is defined as an interval of consecutive control steps that op_i could be scheduled to, denoted as $\mathcal{M}(op_i) = [ms_i, me_i]$, where ms_i is the *earliest* starting time and me_i is the *latest* ending time.

One method to determine mobilities is to use as soon as possible and as late as possible scheduling, as the example shown in Fig.2.3(a). In this case, however, scheduling operations freely within their mobilities may violate data dependencies, for example in Fig.2.3(a) op_1 may be scheduled later than op_2. This is due to *mobility overlaps* among operations, defined as:

DEFINITION 2 Two operations op_i and op_j have *mobility overlap* if there is a data dependency from op_i and op_j, and their mobilities satisfy $\mathcal{M}(op_i) \cap \mathcal{M}(op_j) \neq \emptyset$.

Fig.2.3(c) shows examples of mobility overlaps between op_1 and op_2, op_2 and op_3, and op_3 and op_4. The mobility overlaps are expected to be removed, because: (1) since the existence of mobility overlaps is the essential reason for data dependency violation[22], without mobility overlaps, naturally the data dependencies will not be violate; (2) thus, operations can be scheduled within their mobilities freely; (3) consequently, the calculations of operation probabilities and resource densities are more accurate than the ones with overlaps, which makes the resource usage estimation more accurate.

2.3.3 Motivations for Mobility Graph and Mobility Allocation

For mobility overlap removal, there are several existing works. For example in[22], as shown in Fig.2.3(d), the overlaps are removed operation by operation: once an operation is picked up and mobility overlap is removed, the mobilities of all related operations are updated using a depth first search on DFG.[23] uses a similar way to remove mobility overlaps using Simulated Annealing. These proposals are very time consuming since *mobilities of operations are defined on vertices on the original DFG*, as shown in Fig.2.3(a).

So in this work, *mobilities of operations will be defined on edges on a graph, called mobility graph.* It means, each

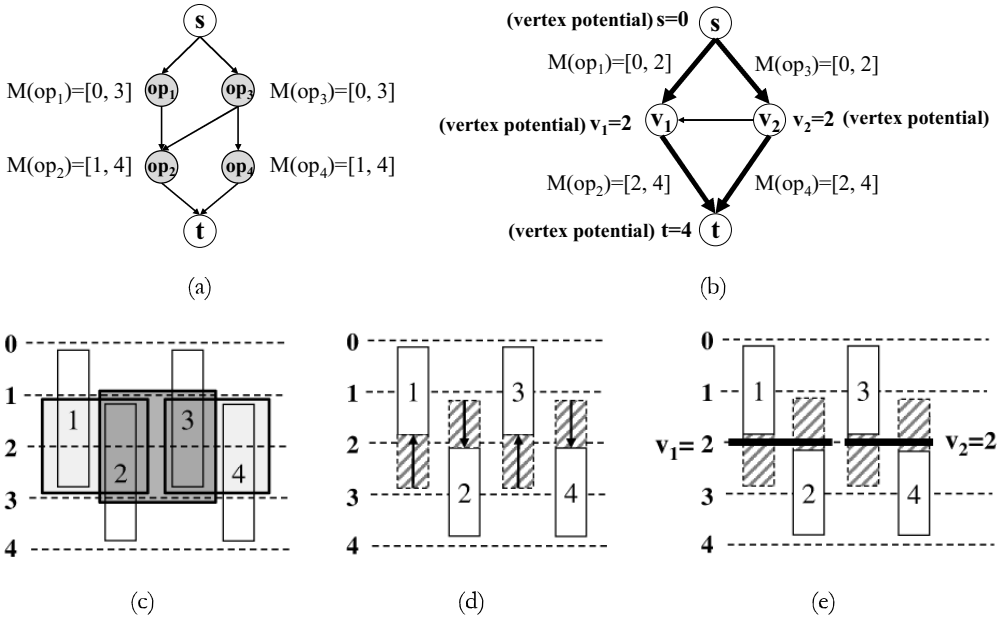


Figure 2.3: Motivations for mobility graph and vertex potential. (a) In previous works mobilities are defined on vertices. ( ) Mobility graph: mobilities are defined on edges. (c) Mobility overlaps. (d) Mobility overlap removal in[23]. (e) Overlap-free mobility allocation in this proposal.

operation is represented by an edge on the mobility graph: the two vertices of the edge refer to time points ,the earliest starting schedule time and latest ending schedule time respectively, which are the mobility boundaries of the operation. An example mobility graph is shown in Fig.2.3(b), corresponding to the DFG in Fig.2.3(a). On this graph, operations are represented by the bold edges, say edge (s, v_1) represents op_1, and (v_1, t) represents op_2. The vertices associated with values (time points) restrict the mobilities of operations; for example $s = 0$, $v =$ $=$ $=$, which means the mobility of op_1 is $\mathcal{M}(op_1) = [0, 2]$, and so as op_2 to op_4. These values are called **vertex potentials**. The detailed definitions of the mobility graph and vertex potential will be given in Section 2.4.3 and Section 2.4.4, respectively.

The set of mobilities of all operations is called a **mobility allocation**. For example given the mobilities shown in Fig.2.3(b), its corresponding mobility allocation is shown in Fig.2.3(e). The labeled lines correspond to the vertices on the mobility graph with vertex potentials. Evidently, it is a *overlap-free mobility allocation*, which is also *dependency-free* among operations. Thus, an overlap-free mobility allocation is obtained from a mobility graph, determined by vertex potentials.

2.3.4 Motivations for Vertex Potential

Vertex potentials proposed in this work not only determine mobilities of operations, but can also solve two sub-problems simultaneously. Examples are shown in Fig.2.4 (original DFG is in Fig.2.3(a)), where all operations are assumed to be the same type, say adder, and the possible delays are 1 and 2.

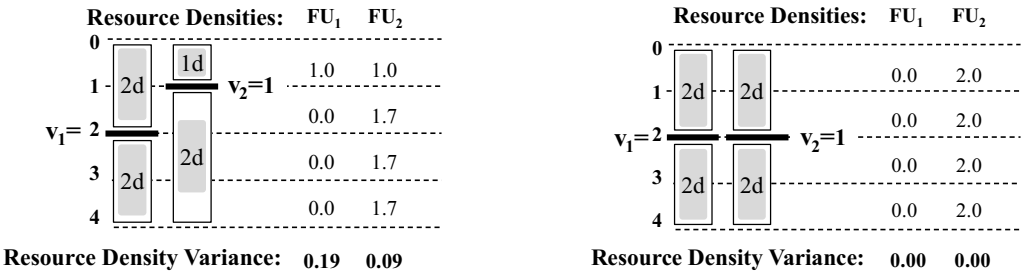


Figure 2.4: Vertex potentials determine both delay assignment and resource density — the common control variable for power and resource optimization. (a) One FU_1 (1-delay) and two FU_2 (2-delay) are needed. (

.

Vertex Potential and Delay Assignment

As shown before, vertex potentials determine a dependency-free mobility allocation, so that operations can be scheduled within their mobilities freely and can be assigned arbitrary delays; to minimize power, each operation should be assigned largest delay within its mobility. Thus, by controlling vertex potential, the sub-problem of delay assignment can be optimized.

Vertex Potential and Resource Density

Given vertex potentials, a , as shown in Fig.2.4(a) and Fig.2.4(b), and resource usage can be approximately estimated. By adjusting vertex potentials, resource densities can be distributed more uniformly. For example in Fig.2.4(b), resource density variances are smaller than those in Fig.2.4(a), which has a larger opportunity to achieve smaller resource usage. Thus, by controlling vertex potential, resource density variances can be minimized.

Therefore, the vertex potentials are the key variables to solve two sub-problems simultaneously. In the following sections, how operation delay assignment and resource density variance minimization are solved by vertex potentials, are discussed in detail in Section 2.4 and Section 2.5 respectively.

2.4 Operation Delay Assignment

In this section the first sub-problem, operation delay assignment is introduced; the second sub-problem, resource density variance minimization, is to be discussed in the next section.

The objective of operation delay assignment is to assign as large as possible delays to operations within latency constraint to minimize P, which is a function of delays of all operations:

$$P = \begin{cases} \sum_{op_i \in V} DY_{op_i}(d(op_i)), & \text{for dynamic power;} \\ \sum_{op_i \in V} LK_{op_i}(d(op_i)), & \text{for leakage power.} \end{cases} \tag{2.11}$$

When P is defined for dynamic power (problem 1), it is the summation of dynamic power of all operations; in this case the delay assignment problem is equivalent to dynamic power minimization, i.e., $min : P = TP_{dy}$. When P is defined for leakage power minimization (problem 2), it is computed as

the summation of leakage power of all operations; it is not exact but a good approximation for overall leakage power, which is introduced in Section 2.8.2.

In this section I discuss the case for dynamic power as an example to show how P is minimized, and is applied for leakage power similarly. In the following, the sub-problem of delay assignment is first relaxed into a linear programming from ILP formulation, then the definitions of mobility graph, mobility allocation and vertex potential are given.

2.4.1 Linear Programming (LP) Relaxation From ILP

The ILP formulation for the sub-problem of delay assignment, including constraints of Eq.2.1 to Eq.2.4 and objective function of Eq.2.7, can be written into a matrix equation $Ax = B$: the coefficient matrix $A = (a_{ij})_{m \times l}$, where $a_{ij} \in \{0, 1, -1\}$, $x = [p_1, q_1, p_2, q_2, \cdots, p_n, q_n, s, t]^T$. Matrix A is node-arc incidence matrix of a network graph, w -1 entry while other are 0; so that A is a *totally unimodular matrix*. Because all the values in A and B are integers, and A is totally unimodular, the equation $Ax = b$ has integer feasible solutions[19]. Accordingly, the ILP formulation for the delay assignment problem can be relaxed to an LP problem which has the same integer feasible solutions with ILP. When the cost function is convex, as the power-delay curve shown in Fig.2.6(a), the optimum solution can be obtained.

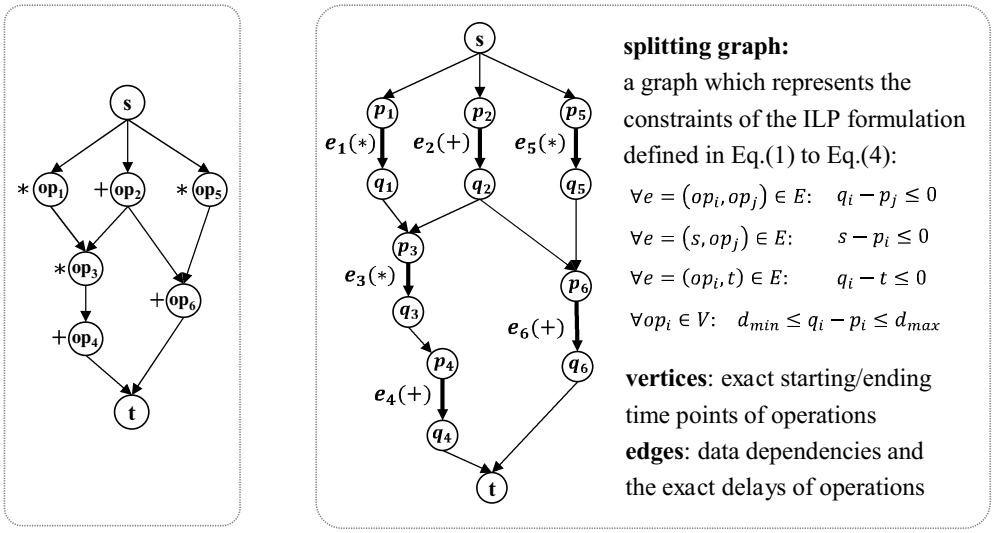
The LP problem is solved using the *piecewise-linear extended Network Simplex Method* (PLNSM)[4]. Since the LP formulation will be solved repeatedly in the scheduling algorithm, its high efficiency is crucial and expected. Especially in this work where the objective function is piecewise-l

, w , a splitting graph is first defined as:

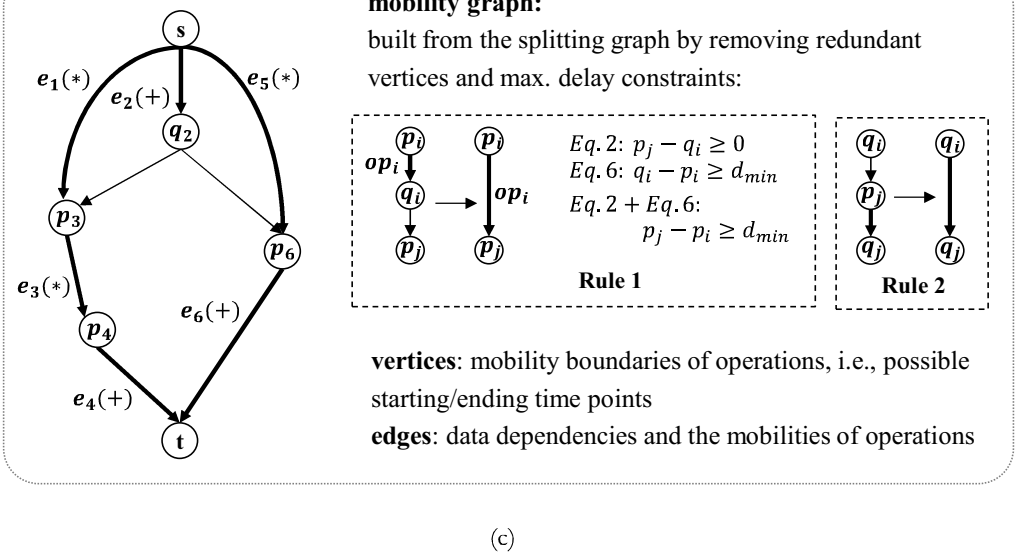
DEFINITION 3 **Splitting graph**, denoted as $G_s = (V_s, E_s)$, is a graph to represent the constraints of the ILP formulation defined in Eq.1 to Eq.4: the vertices represent the variables p_i and q_i of operation op_i, where p_i is the *exact* starting time and q_i is the *exact* ending time of op_i; directed edge $e = (p_i, q_i)$, called *splitting edge*, represents operation op_i, and the edge length l_e is the *exact* delay of op_i, where $l_e = q_i - p_i \in [d_{min}, d_{max}]$. Each edge is associated with a cost $\omega_e = DY\ (l_e)$, referring to the dynamic power of op_i when its delay is l_e.

Fig.2.5(b) shows an example splitting graph, whose original DFG is in Fig.2.5(a). The bold edges are splitting edges representing operations, for example $e_1 = (p_1, q_1)$ represents op_1 in Fig.2.5(a), w

, motivated in Section 2.3.3, is built from the splitting graph with two purposes: (1) to relax the maximum operation delay constraints and execution time points restricted by split edges; (2)



(a) (



(c)

Figure 2.5: (a) DFG. (b) The splitting graph built from (a). (c) The mobility graph built from (b) by redundant vertex removal under two rules.

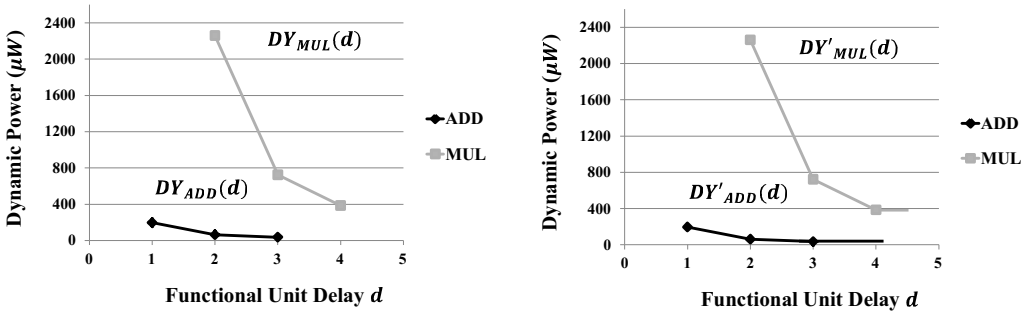


Figure 2.6: (a) Original cost functions for MUL and ADD. (b) Extended cost functions.

to reduce splitting graph size. Both purposes can be achieved by redundant vertex removal and cost function $DY\ (l_e)$ extension associated on split edges.

A vertex on the splitting graph, w -degree and out-degree are both 1, is called a *redundant vertex*. For example on the splitting graph in Fig.2.5(b), vertices like p_1, p_2, q_3, are redundant and can be removed, but vertex p_3 is not because its in-degree is 2. A redundant vertices can be removed because, the *exact* starting and ending execution time points of operations, restricted on the splitting graph, can be relaxed to *possible* time points — the *earliest* starting and *latest* ending execution time points — which is the mobility of operations.

Redundant Vertex Removal

According to Eq.2.1 and Eq.2.4, given two operations op_i and op_j with data dependency, there are: $p_j - q_i \geq 0$ (Eq.2.1) and $q_i - p_i \geq d_{min}$ (Eq.2.4); adding two equations, $p_j - p_i \geq d_{min}^k$ and variable q_i, which is a redundant vertex, is removed. Thus, the mobility of op_i is extended from $[p_i, q_i]$ to $[p_i, p_j]$. Similarly, by adding $p_j - q_i \geq 0$ and $q_j - p_j \geq d_{min}$, p_j is removed, corresponding to Rule 2.

Fig.2.5(c) shows the mobility graph built from splitting graph in Fig.2.5(b) by redundant vertex removal, following Rule 1 and Rule 2. For op_1 represented by e_1 in Fig.2.5(b), by removing the redundant vertices p_1 and q_1, the mobility of op_1 is extended from $[p_1, q_1]$ to $[s, p_3]$. For op_3 represented by e_3, its mobility is extended from $[p_3, q_3]$ to $[p_3, p_4]$.

Cost Function Extension

Since when variable q_i is removed as shown in Rule 1, the maximum delay constraint, $q_i - p_i \leq d_{max}$, written in Eq.2.4, is also removed. To remain the objective function unchanged after removing maximum delay constraint, the cost function $DY\ (d)$ is extended as:

$$DY'\ (d) = \begin{cases} DY\ (d), & d \leq d_{max} \\ DY\ (d_{max}), & d > d_{max} \end{cases} \tag{2.12}$$

$DY'\ (d)$ is the extended cost function from $DY\ (d)$, where Fig.2.6(a) and Fig.2.6(b) show this

extension. When d is smaller or equal to d_{max}, the value of extended DY' equals to original DY (d); when d is larger than d_{max}, the value of DY' (d) shall always be DY (d_{max}), because even d is larger than d_{max}, the possible delay assigned to an operation is at most d_{max}.

Thus, by redundant vertex removal and power-delay function extension, a mobility graph is built from a splitting graph: DEFINITION 4 **Mobility graph**, denoted as $G_m = (V_m, E_m)$, is a graph on which, the mobilities of operations, as well as data dependencies among operations, are represented by edges, and the problem objective function is defined by edge costs.

On the mobility graph, $E_m = E_{m_o} \cup E_{m_d}$, where edge $e_i \in E_{m_o}$ represents operation op_i, and E_{m_d} represents data dependencies. For each $e_i = (u, v) \in E_{m_o}$, the edge length of e_i, denoted as l_{e_i}, is the mobility length of op_i. Similar to splitting graph, each edge is also associated with a cost $\omega'_e = DY'$ (l_e), w

, I now give the exact definitions of vertex potential and mobility allocation in this section.

DEFINITION 5 Given mobility graph $G_m = (V_m, E_m)$, $\forall v_i \in V_m$, v_i is associated with a value, denoted as vp_i, defined as the **vertex potential**. All vertex potentials are written into a vector, denoted as VP. The vertex potentials satisfy:

1. for each edge $e = (u, v) \in E_{m_o}$, $vp_v - vp_u \geq d_{min}^e$;
2. for each edge $e = (u, v) \in E_{m_d}$, $vp_u \leq vp_v$;

DEFINITION 6 A **mobility allocation** is defined as a valid solution set of vertex potentials. For each edge $e = (u, v) \in E_{m_o}$ which represents an operation, its allocated mobility is represented using the vertex potential as $\mathcal{M}(e) = [vp_u, vp_v]$. A valid mobility allocation is denoted as a vector VP to represent vertex potentials for all operations.

Fig.2.7(a) and Fig.2.7(b) show an example of vertex potentials on a mobility graph with its corresponding mobility allocation. In Fig.2.7(a) each vertex v_i on the mobility graph is associated with a vertex potential value vp_i defined in Definition 5. Fig.2.7(b) shows the corresponding mobility allocation derived from Fig.2.7(a). For example, for edge $e_3 = (v_2, v_5)$, where $vp_2 = 4$ and $vp_4 = 7$, its allocated mobility is $[4, 7]$, which means that the execution of operation op_3 must start after time point 4 and ends before time point 7.

PROPERTY 1 **Mobility-overlap-free:** Once a mobility graph is built, the mobility overlap between operations are naturally removed. For any two operations op_i and op_j with allocated mobilities $[ms_i, me_i]$ and $[ms_j, me_j]$, even if there is a data dependency from op_i to op_j on DFG, it is guaranteed that $me_i \leq ms_j$, thus the data dependencies are never violated.

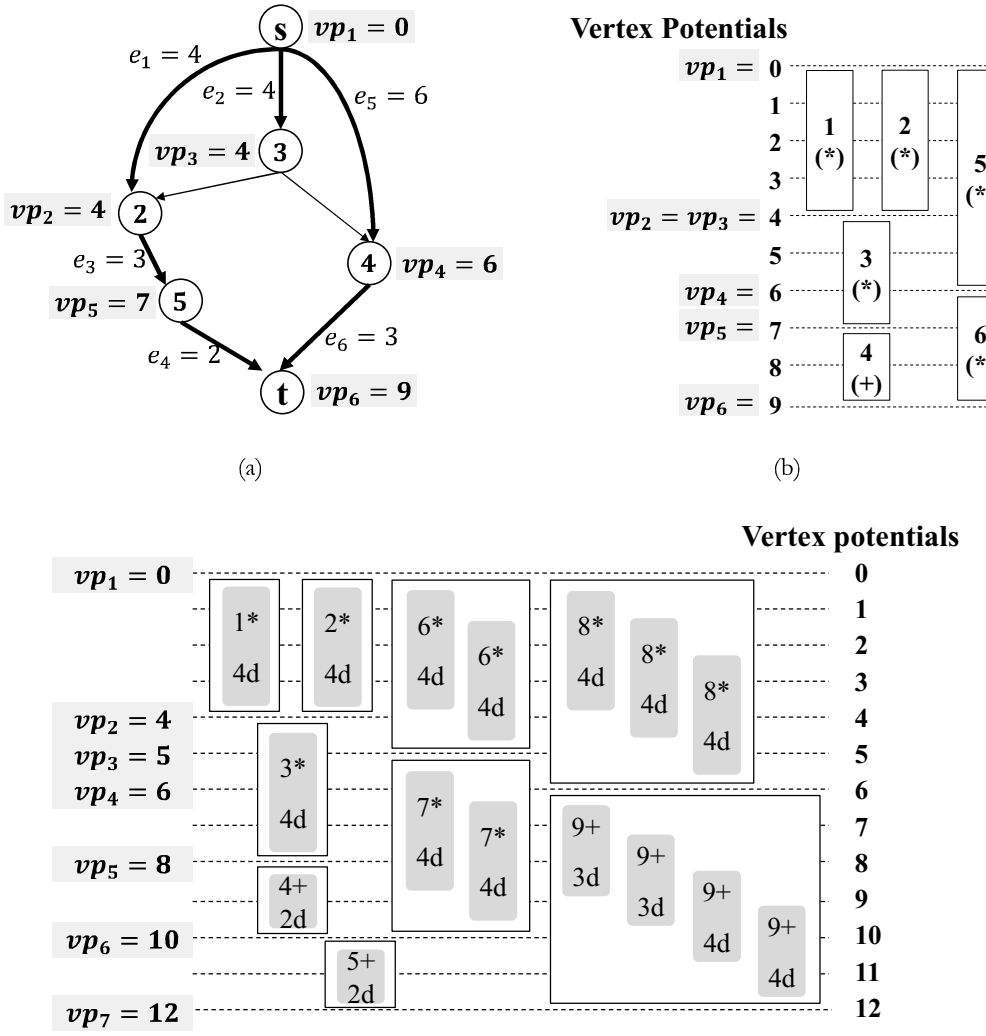


Figure 2.7: (a) The mobility graph with vertex potentials. (. (c) Delay assignment and scheduling

2.4.5 Delay Assignment

Given a mobility allocation, each operation, represented by edge $e = (u, v)$, is assigned as the largest delay within its mobility $[vp_u, vp_v]$ to minimize power consumption:

$$d(e) = \min\{vp_v - vp_u, d_{max}^e\} \tag{2.13}$$

For example as Fig.2.7(c) shows, for the multiplication operation op_6, its largest delay is 4 and allocated mobility is $[0, 5]$, so delay 4 is assigned to op_6; for the addition operation op_4, its largest delay is 3 and allocated mobility is $[8, 10]$, delay 2 is assigned to op_4. Since this assignment always assigns largest allowable delays to operations, it may result in larger numbers of FUs and thus larger leakage power consumption. To eliminate this limitation, a post processing is proposed for leakage power

minimization, which is to be discussed in Section 2.8.2.

2.4.6 Mobility Allocation Problem

Consequently, the sub-problem of delay assignment is transformed to a **mobility allocation** problem as the following:

Given mobility graph $G_m = (V_m, E_m)$, a mobility allocation is to be found, w
$\in V_m$ a vertex potential vp_i is associated that minimizes:

$$min : P = TP_{dy} = \sum_{e=(u,v)\in E_{m_0}} DY'_e(vp_v - vp_u) \tag{2.14}$$

Although the delay assignment problem is discussed by taking dynamic power minimization as an example, it is solved in the same way for leakage power minimization. For simplicity a vector VP is adopted to represent all vertex potentials, and the objective function P in Eq.2.11 is rewrote as:

$$P = \begin{cases} \sum_e DY''(VP), & \text{for dynamic power;} \\ \sum_e LK''(VP), & \text{for leakage power.} \end{cases} \tag{2.15}$$

where e refers to the edges on a mobility graph that represent operations. DY'' is a function of VP where dynamic power data are used, and similarly LK'' is a function of VP where leakage power data are used.

As discussed in Section 2.4.1, the mobility allocation problem is also a linear programming problem, since its coefficient matrix is a node-arc incidence matrix of the mobility graph. The PLNSM is adopted to solve mobility allocation problem.

2.5 Resource Density Variance Minimization

In this section, the second sub-problem, resource density variance minimization, is to be solved simultaneously with the first sub-problem by vertex potentials.

2.5.1 Resource Density

During operation scheduling, it is difficult to directly count the number of FUs; instead, resource density is used to estimate the number of FUs, defined as: DEFINITION 7 The **density of an operation** $e = (u, v) \in E_{m_0}$ with allocated mobility $[vp_u, vp_v]$ at control step t, denoted as dn^e_t, is defined as:

$$dn^e_t = \begin{cases} \frac{d(e)}{vp_v - vp_u}, & vp_u \le t \le vp_v \\ 0, & others \end{cases} \tag{2.16}$$

where $d(e)$ is the delay assigned to operation e in Eq.2.13.

DEFINITION 8 The **resource density of a FU** of type τ and delay d at control step t, is defined as the the summation of densities of all the operations of the same type that is scheduled in t. Since it

is a function of vertex potentials VP, it is denoted as $dn_t^{d,}\quad(VP)$ and calculated as:

$$dn_t^{d,}\quad(VP)=\sum_{op_i\in V} dn_t^{op_i} \tag{2.17}$$

Since the resource usage is greatly affected by the maximum density among all the control steps, the resource densities are expected to distribute uniformly; therefore the *variance* of resource densities is to be minimized:

$$min: R=\sum_{\in\Gamma}\sum_{d_{min}\le d\le d_{max}}\sum_{1\le t\le T_{con}}(dn_t^{d,}\quad(VP)-dn_a^{d,}\quad(VP))^2 \tag{2.18}$$

where the $dn_a^{d,}\quad(VP)$ is the average resource density of all the control steps computed as $dn_a^{d,}\quad(VP)=\sum dn_t^{d,}\quad/T_{con}$.

In the objective function of Eq.2.10, taking Eq.2.18 as the R (for resource density variance minimization) and Eq.2.15 as the P (for delay assignment), Eq.2.10 is written as:

$$\begin{aligned} min:\ & P+\gamma\cdot R \\ &=\sum_e DY''(VP)+\gamma\cdot\sum_d\sum_t(dn_t^{d,}\quad(VP)-dn_a^{d,}\quad(VP))^2 \end{aligned} \tag{2.19}$$

where γ has the same meaning as in Eq.2.10, and is given in Table 2.3 for different problems.

2.5.2 Objective Linearizing Using Target Vertex Potential

Observing Eq.2.19, both terms, the delay assignment term $DY''(VP)$ and the resource density term $dn_t^{d,}\quad(VP)$, are both functions of vertex potentials VP. Consequently, by controlling the values of VP, t , as motivated in previous Section 2.3.4. Moreover, if both terms are linear functions of VP, the PLNSM can be applied to get optimal solutions efficiently.

However, since the second term in Eq.2.19 is quadratic, in order to apply PLNSM, *target vertex potentials* are proposed to deal with the quadratic term in a linear way.

DEFINITION 9 Assume that a set of vertex potentials have already been found, under which Eq.2.18 (the quadratic term in Eq.2.19) is minimized; these vertex potentials, denoted as $VP^*=(vp_1^*,vp_2^*,\cdots,vp_n^*)$, are called **target vertex potentials**.

Given target vertex potential VP^*, a linear formula with absolute value is proposed as:

$$\sum_{e=(u,v)\in E_{m_0}} DY'\ (vp_v-vp_u)+\gamma\times\sum_{v_i\in V_m}|vp_i-vp_i^*| \tag{2.20}$$

where $e=(u,v)$ and v_i refer to edges that represent operations and vertices on the mobility graph respectively; $|vp_i-vp_i^*|$ means that the vertex potential vp_i is expected to be close to its target vp_i^*. γ has the same meaning as in Eq.2.10.

By introducing target vertex potentials, linear formula in Eq.2.20 is solved instead of solving

quadratic formula in Eq.2.19; the two equations are not equivalent but similar to each other, since the second term of Eq.2.19 minimizes Eq.2.18 directly, and the second term of Eq.2.20 tries to get as close as possible to optimal target vertex potentials that minimizes Eq.2.18.

Since it is not able to obtain optimal target vertex potentials in one iteration because of its non-linear property, they are iteratively updated to gradually minimize Eq.2.19. In each iteration, given current target vertex potentials, the partial derivative value for each v_i is computed, and the potentials of k vertices with largest absolute values are updated.

To sum up, under fixed VP^*, optimum solutions for Eq.2.20 can be obtained by PLNSM. That is, given a set of target vertex potentials, the mobility allocation problem defined by Eq.2.20, can be solved optimally. In this way, the operation delay assignment and resource density variance minimization are optimized simultaneously.

2.6 Piecewise-Linear Extended Network Simplex Method

As discussed before, mobility allocation and extended mobility allocation problem are solved using the piecewise-linear extended Network Simplex Method which is introduced in this section. First the outline of PLNSM is introduced, then explain its application on mobility allocation problems.

2.6.1 Outline of PLNSM

General Network Simplex Method is a specific class of Simplex Method which is applied on a directed network graph $G = (V, E)$. For each edge e, it is attached with:

- a function f_e as edge cost
- unit cost function f_e' as edge weight, denoted as $wgt(e)$
- edge length, denoted as $len(e)$
- slack value as the largest distance between current $len(e)$ to its upper or lower bound, denoted as $slk(e)$

Suppose a spanning tree T is constructed on graph G, there are following theorems: **Theorem 1.** The non-basic variables of any basic solution correspond to the tree edges on graph G.

Theorem 2. The condition that a primal feasible solution exists is that, the values of all fundamental loops (FL) are non-positive. Here the value of

fundamental loop means the summation of all the edge weights along the loop direction. **Theorem 3.** The condition that a dual feasible solution exists is that, the value of all fundamental cut sets (FCS) are non-negative. Here the value of fundamental cut set means the summation of all the edge weights along the cut set direction.

Basically, Simplex Method has a mapping to Network Simplex Method as:

- non-basic variables $\{nb_i\} \rightarrow$ tree edges $\{e_t^i\}$;

- basic variables $\{b_i\} \rightarrow$ co-tree edges $\{e_c^i\}$;
- entering variable $nb \rightarrow e_t$ with largest value of $FCS(e_t)$
- leaving variable $b \rightarrow e_c \in FCS(e_t)$ with smallest $slk(e_c)$
- Pivot operation $\rightarrow$ Elementary tree transformation

When the cost function f_e becomes convex separable piecewise-linear (P-L) function f_e^k with increasing breakpoints $\gamma_k^{(h)}$:

$$\cdots \gamma_k^{(-2)} < \gamma_k^{(-1)} < \gamma_k^{(0)} < \gamma_k^{(1)} < \gamma_k^{(2)} < \cdots \tag{2.21}$$

Then f_e^k is specified by a linear function on each interval $[\gamma_k^{(h)}, \gamma_k^{(h+1)}]$. In this case the entering variables and leaving variables may move between two adjacent breakpoints instead of lower bound or upper bound, and current interval of variables may change even if pivot operation is not performed, which means, in Network Simplex Method, the edge weight may changes even if elementary transformation is not performed on the spanning tree.

The detailed PLNSM is given in Algorithm 2.

Algorithm 2: PLNSM(G, edge weight functions)

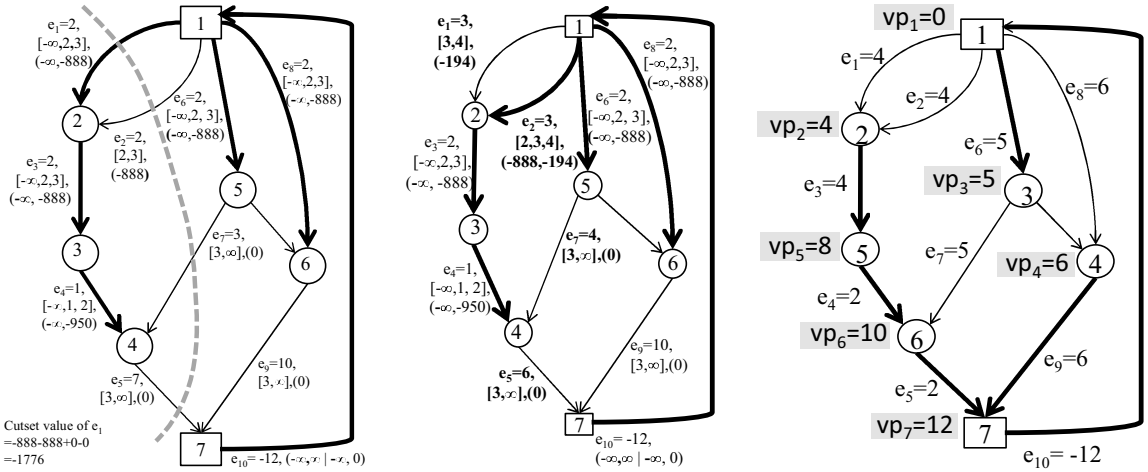
1. *Get initial solution by computing longest path on G,*
   represented by spanning tree T
2. *while(primal feasible solution not obtained) do*
3. *Compute all FCS values for* $e \in T$,
   choose tree edge e_t *with largest FCS value*
4. *Compute* $slk(e)$ *for all edges* $e \in FCS(e_t) \cup \{e_t\}$
5. *if* $e_c \in FCS(e_t)$ *has the smallest slack*
6. *Elementary transformation on T of* e_t *and* e_c
7. *else if* e_t *has the smallest slack*
8. *Tree Structure keeps unchanged*
9. *Update* $wgt(e_c), wgt(e_t), len(e_c), len(e_t)$
10. *endwhile*

In this way, the Network Simplex Method is extended to the PLNSM under piecewise-l

, the PLNSM solves the problem on directed graphs without large matrix operations in traditional Simplex Method, which is supposed to be a much faster specific solver for this problem.

2.6.2 PLNSM Based Mobility Allocation

When applying the PLNSM on mobility graph $G_m = (V_m, E_m)$ where $E_m = E_{m_o} \cup E_{m_d} \cup \{e_T\}$, the $len(e)$ for $e \in E_{m_o}$ means the current mobility of operation e, the $len(e_T)$ means the current control



(a) Initial tree (b) After one transformation (c) Final tree

Figure 2.8: Example of PLNSM on mobility graph.

step number. Further, for $e = (v_i, v_j)$, $len(e)$ is calculated as $len(e) = vp(v_j) - vp(v_i)$; for $e \in E_{m_o}$, $wgt(e)$ is calculated as the interval gradient of $P_{type(e)}$; for $e \in E_d$, $wgt(e) = 0$; for $e = e_T$, $wgt(e) = 0$ if $len(e) \leq T_{con}$ or $wgt(e) = +\infty$ if $len(e) > T_{con}$. Note that $wgt(e)$ is related to the current piecewise intervals of $len(e)$.

Based on the graph G_m, the PLNSM is performed following the Algorithm 2. Fig.2.8 shows an example. In Fig.2.8(a), the bold edges represent tree edges, the value of e_i represents the edge length, the values in square brackets represent the current piecewise intervals of variables, and the values in parenthesis represent the edge weights for each piecewise interval.

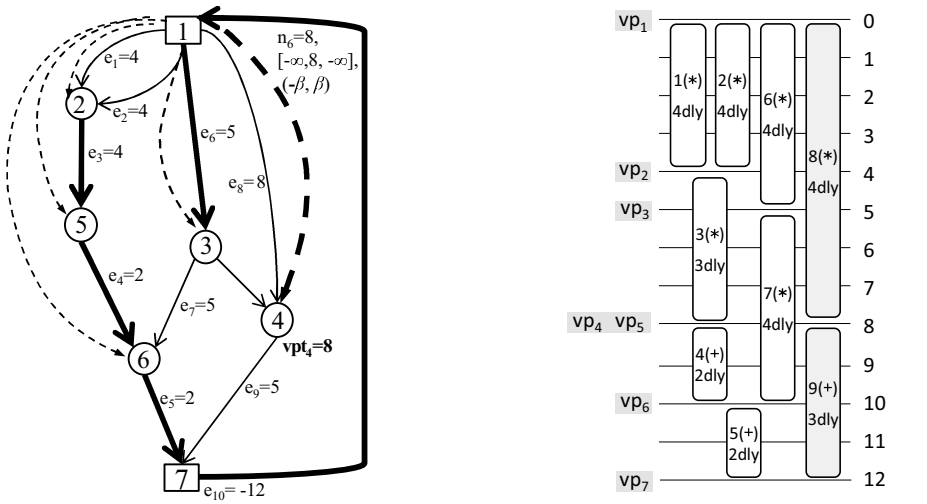
Step 1: Set the minimum $len(e)$ for each e as the initial solution, and build a spanning tree by computing longest path on graph G_m. Note that if there's any positive cycle of fundamental cut sets, it means no feasible solution can be obtained.

Step 2: The PLNSM algorithm terminates when the values of all fundamental cut sets are non-negative, according to Theorem 3.

Step 3: For each tree edge e_t, compute the value of $FCS(e_t)$ as $wgt(e_t) + \sum_{e_c \in FCS(e_t)} \pm wgt(e_c)$, and choose the edge with largest value. Note that the sign of $wgt(e_c)$ is positive if the direction of e_c is along with the cue set, otherwise is negative. For example, for edge e_1, the $FCS(e_1) = \{e_2, e_7, e_5\}$, and $val(e_1)$ is $(-888) + (-888) + (0) - (0) = -1776$.

Step 4-8: Suppose e_t^k is chosen in Step 3, then for each edge $e \in FCS(e_t^k) \cup \{e_t^k\}$, compute $slk(e)$ and choose one with smallest $slk(e)$. In this case, $e_t^k = e_1$ and compute the slack value for $\{e_1, e_2, e_5, e_7\}$ and get $min\{slk(e_1), slk(e_2), slk(e_5), slk(e_7)\} = min\{3-2, 3-2, \infty-3, 5-3\} = slk(e_2)$. So e_2 is chosen and tree transformation is performed by taking e_1 out of the tree and adding e_2 into the tree, as shown in Fig.2.8(b).

Step 9: Update the edge length and weight for e_t and $e \in FCS(e_t)$ according to their new piecewise intervals. In this case, e_1, e_2, e_7 and e_5 is updated as shown in Fig.2.8(b). Since e_1 becomes a co-tree



(a) Final tree with target potential ( ) Corresponding mobility allocation

Figure 2.9: Example of PLNSM with target vertex potential.

edge, its interval reduces to one ($[3, 4]$) and the intervals of e_2 increase to two ($[2, 3, 4]$).

By applying the PLNSM, the final solution is obtained presented by the tree as shown in Fig.2.8(c). The mobilities of operations are allocated along with the vertex potential vector VP be determined.

2.6.3 PLNSM based Extended Mobility Allocation

Given the modified mobility graph, the PLNSM can also apply on it with only $|V|$ extra potential edges.

Fig.2.9 shows an example of the modified mobility graph with final mobility allocation and delay assignment solution. In Fig.2.9(a), the dotted edges represent the target potential edges. In this example, the vertex v_6 is associated with target potential $vp_t(v_6) = 8$ and $\gamma = 1$. As a result, the vertex potential of $vp(v_6)$ moves from 4 to 8 in Fig.2.9(b) due to the introduction of target potential. Fig.2.9(a) shows the final tree and Fig.2.9(b) shows the corresponding mobility allocation.

2.6.4 Analysis of PLNSM

Complexity Analysis

For linear programming problem, the complexity of general Simplex Method as well as the Network Simplex Method are both closely related to the variants and the number of pivot operations. For one particular problem, the Simplex Method and the Network Simplex Method may have the same pivot number but absolutely different complexity of each pivot operation. Especially when the linear programming is extended with a piecewise-linear objective function, the complexity of traditional Simplex Method grows significantly due to the introduction of large amount of new variables to reformulate it as a new linear programming problem[13], while the Network Simplex Method is proved to be scalable with the complexity almost unchanged.

In the algorithm of NPMVS, the kernel step is solving the mobility graph iteratively by Network Simplex Method, whereas the efficiency of Network Simplex Method is crucial to the global efficiency.

Non-convex function Extension

In this work the MVS problem has a convex objective function, which guarantees the optimality of the Network Simplex Method. However even in the case of non-convex power-delay curve, the proposed method can also be applied with a high probability to gain optimum solutions by repeating random starts combined with local search that is able to escape from local optimal solutions.

2.7 Proposed Unified Scheduling Scheme

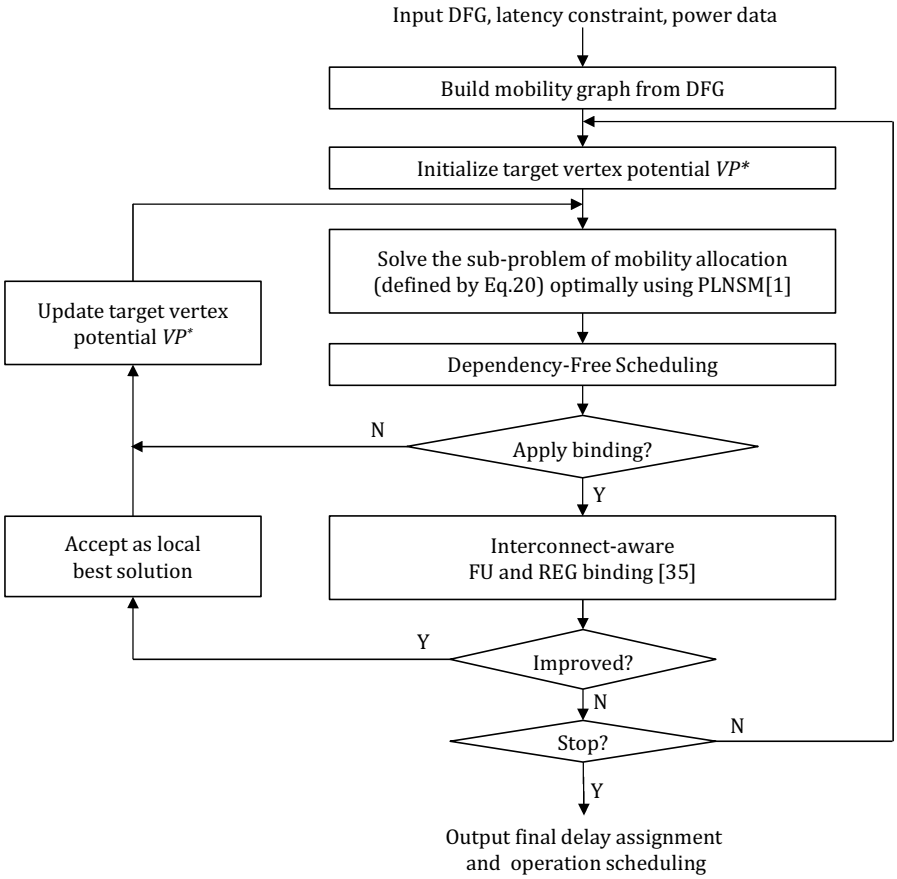


Figure 2.10: Flow-chart of the unified scheduling approach.

In the previous two sections, the two sub-problems are respectively discussed. In this section, the proposed unified scheduling scheme is introduced. Its utilizations on three problems are discussed in Section 2.8.

Algorithm 1 Proposed Unified Scheduling Scheme

Require: Data Flow Graph, latency constraint

Ensure: A Delay assignment $d(op_i)$ and an operation scheduling $s(op_i)$ for each operation op_i

1: Extended mobility graph generation from input data flow graph
2: Initialize target vertex potential vp_i^* for each v_i
3: **while** stop criteria NOT met **do**
4: Update the **vertex potential** for each vertex to minimize Eq.2.20
5: **Dependency-free scheduling**
6: Compute the total power P_{fu}^{curr} of all allocated FUs
7: **if** $P_{fu}^{curr} \leq (1+\delta) \cdot P_{fu}^{curr\_bst}$ **do**
8: Perform FU and register binding
9: Update **target vertex potentials** of k vertices
10: **end while**

2.7.1 Overview of Unified Scheduling Scheme

The overview of the proposed unified scheduling scheme is shown in Algorithm 1, and the flowchart is shown in Fig.2.10.

From input DFG and power data, an extended mobility graph is first built as shown line 1; then the delay assignment problem is first solved without considering resource, i.e., Eq.2.14 is solved optimally using PLNSM; the vertex potentials obtained here are set to be initial target vertex potentials VP^*.

Given initial vertex potentials, the scheduling algorithm is executed iteratively, as shown in line 4-9. First, the mobility allocation defined by Eq.20 is solved using PLNSM, and the optimum vertex potentials with respect to the given target vertex potentials are obtained as line 4. Based on the mobility allocation determined by vertex potentials, dependency-free scheduling is applied as line 5. After dependency-free scheduling, the number of allocated FUs are obtained. dynamic power, leakage power and resource usage evaluated by Eq.2.7 to Eq.2.9, respectively, according to different objectives of the three problems. Then the FU and register binding is performed conditionally, which is to be introduced in Section 2.7.3. Finally, k target vertex potentials are updated as line 9, and extended mobility allocation is solved again under new VP^*. According to the experiments, k is set as 5% to 10% of total vertices on the mobility graph, which shows highest efficiency. The algorithm terminates when the total number of iterations exceeds the preset value.

When the scheduling scheme is applied on problem 3 given two objectives, to obtain Pareto Solutions, a solution is saved either dynamic or leakage power reduces.

2.7.2 Dependency-Free Scheduling

Given an extended mobility allocation, a dependency-free scheduling is proposed to determine the final schedule for each operation, as shown in Algorithm 2. There are two stages: stage 1, shown in line 1 to 6, is to estimate the numbers of FUs for each type τ with delay d, and stage 2, shown in line 7 to 11, is to gradually reduce the estimated numbers of FUs. In stage 1, initially the numbers of all used FUs

Algorithm 2 Dependency-free Scheduling

Require: Extended Mobility allocation with vertex potentials
Ensure: Schedule for each operation
1: *// Stage 1: estimate the number of FUs of each type*
2: Set all numbers of used FU of type τ and delay d as 1
3: **while** Not all operations are scheduled **do**
4: Increase the number of FU by 1 for un-scheduled operation op_i of type τ and delay d
5: Apply list scheduling within current number of FUs
6: **end while**
7: *// Stage 2: reduce the number of FUs*
8: **while** List scheduling succeeds **do**
9: Reduce the number of FU of type τ and delay d by 1
10: Apply list scheduling under decreased FU usage
11: **end while**

of type τ and delay d are set to be 1, and operations are scheduled using list scheduling; the priority is the acceding order of ending times of operations mobilities, and operations shall be scheduled within their mobilities. If op_i is failed to be scheduled within the current number of FUs, the number of FU for op_i is increased by 1, and list scheduling is performed again, until all operations are scheduled. In stage 2, the number of FUs of each type by is tried to reduce by 1 in each iteration; during this step, mobility constraints of operations are relaxed, and list scheduling is applied on the original DFG; the priority is the same as in step 1. The algorithm terminates when the numbers of FUs cannot be further reduced.

2.7.3 Functional Unit and Register Binding

Because FU and register binding step has a large impact on the interconnection complexity and multiplexer sizes, in the proposed scheduling approach, interconnect-aware FU and register binding is also conducted to reduce overall power consumption, including multiplexer power and register power, as shown in line 6. The adopted binding algorithm is proposed in[27], whose objective is to reduce the number of registers and input counts of multiplexers. It is a simplified weighted compatibility graph based dynamic programming method improved from work in[32]. Since performing binding in each iteration is time consuming, it is skipped if the current scheduling solution has a low possibility to reduce the overall power of the current best solution (the scheduling that has lowest overall power consumption up to now): denote the FU power of the current solution as P_{fu}^{curr} and the FU power of the current best solution as $P_{fu}^{curr\_bst}$, the FU and register binding is only performed when $P_{fu}^{curr} \leq (1 + \delta) \cdot P_{fu}^{curr\_bst}$, as line 6-8. In this work δ is set as 0.1. After binding, the numbers of used FUs, registers and multiplexers are known, and the overall power is computed; if it is smaller than the current lowest power, the current scheduling and binding are accepted and P_{fu}^{glob} is updated.

Table 2.3: Scheduling Algorithm Utilizations on Three Problems.

Algorithm Utilizations and Parameters		
Common Objective:	P	$+ \quad \gamma \cdot R$
Problem 1	Descrp.	Dynamic Power Min. + Resource Min.
	Data	Power-Delay Table 2.1 under multi-V_{dd}
	Obj.	$\sum DY''(VP) + \gamma \cdot \sum\sum\sum (dn_t(VP) - dn_a(VP))^2$
	Param.	$\gamma = 0.1$
Problem 2	Descrp.	Leakage Power Min.
	Data	Power-Delay Table 4.1 under multi-V_{th}
	Obj.	$\sum \frac{LK''(VP)}{\varepsilon} + \gamma \cdot \sum\sum\sum LK'(VP) \cdot (dn_t(VP) - dn_a(VP))^2$
	Param.	$\gamma = 5$
Problem 3*	Descrp.	Dynamic Power Min. + Leakage Power Min.
	Data	Power-Delay Table* under multi-V_{dd} and multi-V_{th}
	Obj.	$\sum DY''(VP) + \gamma \cdot \sum\sum\sum LK'(VP) \cdot (dn_t(VP) - dn_a(VP))^2$
	Param.	γ varies for Pareto Solutions

* Problem 3 can only be solved with conditions; in experiments the dynamic power in Fig.10 are from Table II and leakage power are from Table I.

2.8 Algorithm Utilization on Three Problems

In this section, the proposed scheduling scheme is utilized on three problems with different power data, objective functions and parameters, as illustrated in Table 2.3.

2.8.1 Problem 1: Dynamic Power and Resource Co-Optimization

As discussed in Section 2.2.2, only dynamic power minimization can be relaxed to linear programming and solved optimally. When dynamic power is co-optimized with resource usage, solutions with optimal dynamic power values are firstly chosen, and then resource is minimized without sacrificing dynamic power. For example the solutions in Fig.2.1(b) and Fig.2.1(c) have same optimal dynamic power but Fig.2.1(c) has a smaller resource usage. Therefore, its objective function, where P is defined as Eq.2.15 and R is defined as Eq.2.18, is:

$$\begin{aligned} min: \ & P + \gamma \cdot R \\ & = \sum_e DY''(VP) + \gamma \cdot \sum_d \sum_t (dn_t^{d,\ \ }(VP) - dn_a^{d,\ \ }(VP))^2 \end{aligned} \tag{2.22}$$

where in the experiments, γ is set as 0.1, which is a small enough value to suggest that dynamic power is first optimized and then followed by the resource usage.

2.8.2 Problem 2: Leakage Power Minimization

Leakage power minimization also contains the same two sub-problems P and R, as illustrated in Section 2.3.1. Since the estimation of leakage power is different from dynamic power, the definitions of P and R are needed some modifications.

Delay Assignment

As defined in Eq.2.15, P is written as:

$$min : P = \sum_{op_i \in V} LK'_{op_i}(d(op_i)) = \sum_{e} LK''(VP) \tag{2.23}$$

where P is computed as the summation of leakage power of *all operations in DFG*.

On the other hand, as defined in Eq.2.8, the actual leakage power consumption shall be computed as the summation of leakage power of *all allocated FUs* as:

$$P_{actual} = \sum_{\in \Gamma} \sum_{d_{min} \leq d \leq d_{max}} LK'_{\tau}(d) \cdot N(d, \tau) \tag{2.24}$$

Note that P computed in Eq.2.23 is the objective function of operation delay assignment to be minimized, while P_{actual} computed in Eq.2.24 is the actual leakage power consumption. Obviously, the more similar P and P_{actual} are, the more accurate scheduling approach is for leakage power optimization.

To compensate the difference between P and P_{actual}, the operation *reuse ratio*, denoted as ε^d , is proposed to approximately suggest that how many *operations in DFG* can share one physical *allocated FU*. Therefore, dividing P in Eq.2.23 by ε^d for each kind of FU of type τ and delay d, a compensated function P_{compst} is defined as:

$$min : P_{compst} = \sum_{op_i \in V} \frac{LK'_{op_i}(d(op_i))}{\varepsilon^{d(op_i)}_{op_i}} = \sum_{e} \frac{LK''(VP)}{\varepsilon^d} \tag{2.25}$$

P_{compst} is closer to P_{actual} than P, and is the new objective function of operation delay assignment for leakage power minimization. The initial reuse ratios are obtained from pre-experiments for each testbench, by applying a simple list scheduling; during the proposed scheduling algorithm, the ratios are updated as the actual values obtained from previous iteration.

Table IV shows two sets of solutions, where the first ones are obtained by just minimizing function P_{compst} defined as Eq.2.25, and second ones are obtained by minimizing both P_{compst} and R, where R is to be introduced in Section 2.8.2. It shows that, for most cases, the number of FUs obtained by just minimizing P_{compst} is only 1 or 2 more than that both P_{compst} and R are minimized. This implies that, first, the function P_{compst} in Eq.2.25 is a good approximation of leakage power, and thus is effective in minimizing leakage power. Second, the resource minimization cannot be neglected, because a low threshold voltage (*lvt*) FU consumes much more power than a high threshold voltage (*hvt*) FU, and one less *lvt* FU may result in much lower total leakage power. For example for case *rand0* under latency $T_{con} = 25$, the number of *lvt* MUL is reduced by only one if R is considered, but the leakage power greatly reduces from $590.1\mu W$ to $319.4\mu W$. This implies that minimizing R must be conducted for leakage power minimization.

Table 2.4: Leakage power minimization without and with resource concern

DFG	T_{con}	**Only Minimizing P_{compst}**					**Minimizing P_{compst} and R**				
		ADD		MUL		P_{ly}	ADD		MUL		P_{ly}
		hvt	*lvt*	*hvt*	*lvt*	(μW)	*hvt*	*lvt*	*hvt*	*lvt*	(μW)
diff	7	1	1	4	1	547.5	1	1	3	1	520.1
	9	1	1	4	1	276.8	1	1	3	0	249.4
	11	2	0	3	0	96.4	2	0	2	0	69.0
fft	24	12	2	8	1	895.3	6	3	6	0	687.2
	30	12	1	6	0	409.8	9	1	5	0	361.0
	36	8	1	5	0	353.9	6	1	4	0	201.7
rand0	20	8	2	5	1	784.7	7	2	4	1	750.2
	25	7	1	4	1	590.1	7	1	4	0	319.4
	30	6	0	3	0	124.9	6	0	3	0	124.9
rand1	23	13	3	4	1	952.9	12	3	4	0	675.1
	28	12	2	4	0	515.0	11	1	4	0	347.9
	33	11	1	3	0	320.5	9	1	3	0	306.2
rand2	36	46	3	14	1	1461.5	40	3	13	1	1391.5
	45	33	2	13	0	910.9	32	1	10	0	661.6
	54	28	1	9	0	605.7	26	0	8	0	404.1

R

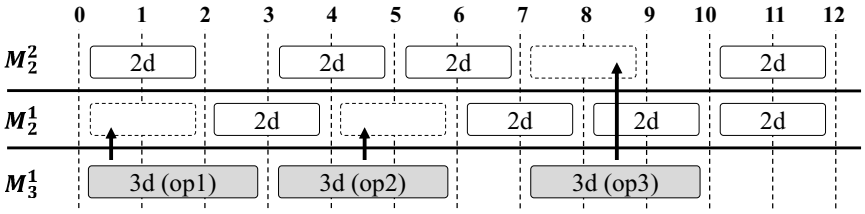
, when R is computed for leakage power, the weight for the density of FUs of type τ and delays d shall be set as its leakage power consumption $LK\ \ (d)$, to estimate actual total leakage power more accurately, because the resource density is positively related to actual number of FUs. Besides, although the computation for R is an approximation for leakage power, the number of FUs and leakage power are computed according to Eq.2.8 each time after dependency-free scheduling, to evaluate actual leakage power more accurately. Consequently for leakage power minimization, R is computed as:

$$R = \sum_{d} \sum_{t} LK'\ \ (d) \cdot (dn_t^{d,}\ \ (VP) - dn_a^{d,}\ \ (VP))^2 \tag{2.26}$$

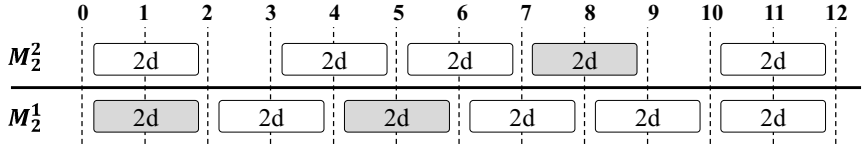
To sum up, the objective function for leakage power minimization, is written as:

$$\begin{aligned} min : & \sum_{e} \frac{LK''(VP)}{\varepsilon^d} + \\ & \gamma \cdot \sum_{d} \sum_{t} LK'\ \ (d) \cdot (dn_t^{d,}\ \ (VP) - dn_a^{d,}\ \ (VP))^2 \end{aligned} \tag{2.27}$$

where γ is set as larger than that in problem 1, say $\gamma = 5$ in experiments because for problem 2, the numbers of FUs primarily determine the leakage power and thus need to be minimized under a large weight. It is also solved on the mobility graph using PLNSM. Eq.2.27 is linearized in a similar way as Eq.2.20 being linearized from Eq.2.19.



(a) FU allocation before FU mergence, where three FUs are used.



( ) operations are re-assigned to smaller delays to reduce the number of FUs, where only two FUs are used.

Figure 2.11: Example of post processing for leakage power minimization.

Post Processing

As discussed in Section 2.4.5, in this proposal, given a mobility allocation, to each operation the largest possible delay is always assigned, which may potentially increase the number of FUs of larger delays, and thus increase total leakage power. For example as a solution produced by this scheduling approach shown in Fig.2.11, three FUs are used, two 2-delay multipliers and one 3-delay multiplier. However if the three 3-delay operations are reassigned as 2-delay, all of them can reuse the already allocated 2-delay multipliers and the 3-delay multiplier can be saved.

Consequently, after dependency-free scheduling, a post processing for leakage power minimization is proposed: by reassigning smaller delays to operations and rescheduling these operations to other FUs, the number of FUs can be reduced.

For each FU of type τ and delay d, each operation op_i implemented on it is checked whether op_i can be reassigned to delay d' that $d' \leq d$ and rescheduled to other FUs of delay d'. For example for M_3^1 in Fig.2.11(a), op_1, op_2 and op_3 are checked. op_i can only be rescheduled within $[t_i,\ t_i + d]$, where t_i is the current starting control step of op_i, in order to guarantee that data dependencies are not violated. For example for op_3 in Fig.2.11(a) whose original schedule is $[7, 10]$, it can be rescheduled to $[7, 9]$ or $[8, 10]$, as shown in Fig.2.11(b). If all operations on a FU can be rescheduled by being reassigned smaller delays, this FU is saved, as shown in Fig.2.11(b), M_3^1 is saved and only two FUs are used.

2.8.3 Problem 3: Dynamic and Leakage Power Co-Optimization

Problem 3 has a major difference from problem 1 and 2. In problem 1 and 2, either of the voltages V_{dd} or V_{th} is fixed; then, once the delay of an operation is determined, the other voltage, V_{th} or V_{dd}, is uniquely determined. Meanwhile, the leakage power or dynamic power of the FU with the determined delay is also unique. Therefore, the scheduling approach deals with operation delays instead of real voltage values. On the other hand, in problem 3, multi-V_{dd} and multi-V_{th} are given at the same time.

In this case, if the pair of V_{dd} and V_{th} still can be uniquely determined by operation delay, this unified scheduling approach can be applied. Under this assumption, the objective function of problem 3 is written as:

$$min : \sum_{e} DY''(VP) + \gamma \cdot \sum \sum_{d} \sum_{t} LK \quad (d) \cdot (dn_t^{d,} \quad (VP) - dn_a^{d,} \quad (VP))^2 \tag{2.28}$$

where P is defined as Eq.2.15 for dynamic power minimization, R is defined as Eq.2.26 for leakage power minimization. γ is used to balance between dynamic and leakage power, whose values are given in Section 2.9.3.

Since there are two objectives of problem 3, the dynamic power and the leakage power, a unique optimal solution may not exist. Therefore, a function is added in Algorithm 1, to save the Pareto Solutions during the minimization of Eq.2.28.

On the other hand, if one operation delay corresponds to more than one pairs of V_{dd} and V_{th}, an additional processing is necessary to select one pair from the different pairs, w

(, , , 50, 100, 200, 250), first as 5, and is increased to the next larger value until the scheduling algorithm terminates.

The results show that for all benchmarks that ILP succeeded, this scheduling approach is able to produce solutions with both optimum dynamic power and resource usage the same as ILP solutions. Further, among 22 groups of data the longest running time of the algorithm is 0.25sec (FFT under

$T_{con} = 36$), and the average run time is less than 0.1sec.

Table 2.5: Comparisons between this scheduling approach and ILP solution

DFG	T_{con}	MINE								ILP						
		# of itr.	P_{dy} (μW)	4d*	3d*	2d*	3d+	2d+	1d+	P_{dy} (μW)	4d*	3d*	2d*	3d+	2d+	1d+
ad2	47	5	68664	1	2	3	0	1	2	68664	1	2	3	0	1	2
	70	5	23463	2	3	1	2	0	0	23463	2	3	1	2	0	0
	94	5	15385	3	1	0	2	1	0	15385	3	1	0	2	1	0
ae	72	10	52747	2	1	2	1	1	2	52747	2	1	2	1	1	2
	90	50	26815	1	2	1	1	2	0	NA	NA	-	-	-	-	
	108	50	20374	3	1	0	2	1	0	NA	NA	-	-	-	-	
ar	13	50	18380	4	0	4	1	0	2	18380	4	0	4	1	0	2
	16	100	9524	2	4	0	1	2	2	9524	2	4	0	1	2	2
	19	100	7636	4	0	0	1	2	0	7636	4	0	0	1	2	0
diff	7	50	6030	1	1	2	1	1	1	6030	1	1	2	1	1	1
	9	50	3138	2	2	0	1	1	1	3138	2	2	0	1	1	1
	11	100	2666	3	0	0	1	1	0	2666	3	0	0	1	1	0
ellip	25	20	47850	1	1	3	0	2	2	47850	1	1	3	0	2	2
	30	20	29777	2	2	1	1	2	0	29777	2	2	1	1	2	0
	35	50	10925	4	1	0	2	0	0	10925	4	1	0	2	0	0
mpeg	36	5	16451	1	0	1	1	1	2	16451	1	0	1	1	1	2
	42	50	10826	1	0	1	1	2	1	NA	NA	-	-	-	-	
	50	50	8697	1	1	0	2	2	0	NA	NA	-	-	-	-	
fft	24	100	22329	4	6	2	6	8	4	22329	4	6	2	6	8	4
	30	100	17516	5	4	1	9	5	1	NA	NA	-	-	-	-	
	36	100	15841	7	2	0	10	2	0	NA	NA	-	-	-	-	
AVG			**1.0**							**1.0**						

2.9.2 Leakage Power Minimization

In this section the proposed scheduling algorithm is applied for leakage power minimization, by comparing it to the latest existing work BCGS[23] and an early work MWIS[24], as shown in Table 3.4. To make a comparison, the same formulation is adopted as in[23], where dual threshold voltages are used: high-V_{th}, denoted as *hvt* and low-V_{th}, denoted as *lvt*.

In terms of leakage power consumption, first, compare MINE and BCGS to MWIS showing in the row of AVG1. It shows that BCGS reduces leakage power by 52.0% compared to MWIS, while my proposal MINE reduces leakage power by 61.8% compared to MWIS. Secondly, MINE is compared to BCGS, showing in the row of AVG2. It shows that my proposal reduces leakage power by 20.4% compared to BCGS. Moreover, comparing to small testcases, my proposal saves much more power under larger cases, say *fft*, *rand*0, *rand*1 and *rand*2; under these four cases my approach saves 43.9% leakage power on average compared to BCGS[23].

In terms of execution time, BCGS suffers from 2.0 times overhead compared to MWIS.† Comparing to MWIS, the CPU time of my proposal is only 3.8% of MWIS, which is a 26X speedup;

†The authors of BCGS adjust the parameters to let their algorithm produce solutions as good as possible, say by increasing the iteration time, ect., so that for fairness the two cases are excluded when comparing the CPU time.

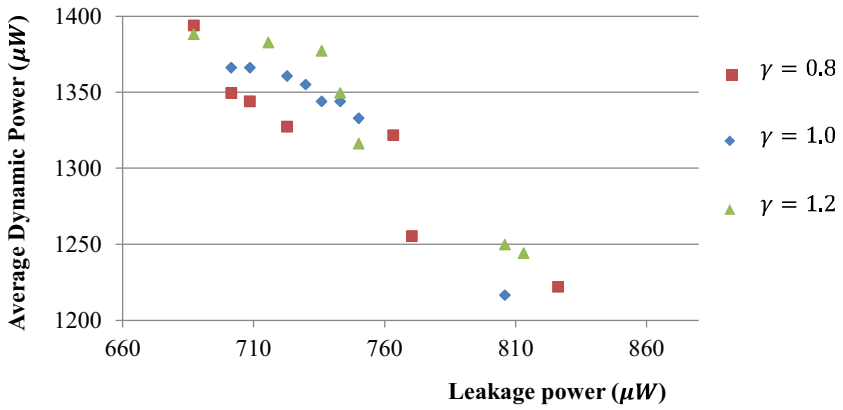


Figure 2.12: Dynamic power and leakage power solution space.

and comparing to BCGS, the CPU time of my proposal is 1.9% of BCGS, which is a 52X speedup. For test bench *rand2*, the actual comparison is less than $1.0e-4$ but for simplicity it is recorded as 0.001.

The experimental results imply that my approach is efficient for leakage power minimization, where both power and execution time excel[23]. The reasons are analyzed as:

- For power consumption, in[23], all operations are initially assigned by smallest delays and reassigned to larger delays during the optimization; thus, the initial solution is farthest from final solution, and the optimization is easily fallen into local optimum solutions. On the other hand, in my proposal, the initial solution is computed using linear programming, which is quite close to final solution and can be obtained in one iteration. For example as shown in Table IV, only minimizing P can already get solutions quite close to the final solution; based on a good initial solution, R is optimized to further reduce leakage power.
- For execution time, in[23], the most time consuming part is the perturbation of mobility overlap removal, where Simulated Annealing is adopted; while in my proposal, mobility-overlap-free solutions are easily obtained by adjusting vertex potentials, w

-Optimization

To show concept of problem 3, the possible delays for adders are assumed as (1,2) and for multipliers are (2,3); dynamic power in Table 2.1 and leakage power in Table 4.1 are adopted. Because there are two optimization objectives, the leakage power and dynamic power, Pareto Solutions are shown in Fig.2.12 under testbench FFT with latency $T_{con} = 24$. The power values are obtained under different parameters $\gamma = 0.8$, $\gamma = 1.0$ and $\gamma = 1.2$. In order to compare leakage and dynamic power, the dynamic power shown here is the average consumption during all control steps. The overall trend shows that along with the decreasing of dynamic power, the leakage power increases, which is typical for Pareto Solutions. On the other hand, with the decreasing of dynamic power, operations are assigned

Table 2.6: Comparisons between my scheduling approach and existing work[23] for leakage power minimization only.

DFG	T_{con}	MINE									BCGS[23]								MWIS[24]					
		# of itr.	ADD *hvt*	ADD *lvt*	MUL *hvt*	MUL *lvt*	P_{ly} (μW)	Cmp	t (ms)	Cmp	ADD *hvt*	ADD *lvt*	MUL *hvt*	MUL *lvt*	P_{ly} (μW)	Cmp	t (ms)	Cmp	ADD *hvt*	ADD *lvt*	MUL *hvt*	MUL *lvt*	P_{ly} (μW)	t (ms)
ad2	47	5	0	2	0	3	1132.2	1.000	0.70	0.017	0	2	0	3	1132.2	1.000	7400	176^{+}	0	2	0	3	1132.2	42
	70	10	0	1	3	0	242.3	0.273	1.30	0.022	0	1	1	1	458.2	0.515	190200	3279	0	2	1	2	888.9	58
	94	10	1	0	2	0	61.9	0.248	1.71	0.035	1	0	2	0	61.9	0.248	303600	6195	1	1	3	0	249.4	49
ae	72	10	0	1	1	1	458.2	0.532	2.62	0.006	2	0	3	1	367.1	0.426	1770	3.91	0	2	0	2	861.5	453
	90	200	1	1	3	0	249.4	0.544	46.42	0.112	1	1	3	0	249.4	0.544	1902	4.57	0	1	1	1	458.2	416
	108	100	1	0	2	0	61.9	0.241	18.31	0.046	3	0	2	0	76.1	0.297	3954	9.89	2	1	3	0	256.5	400
ar	13	50	0	2	4	2	971.1	0.611	0.52	0.005	0	2	0	4	1402.9	0.882	239	2.08	0	3	1	4	1590.4	115
	16	200	0	2	4	0	429.7	0.285	15.96	0.126	0	2	5	0	457.1	0.303	402	2.98	0	4	2	3	1507.1	135
	19	200	2	0	4	0	123.8	0.154	12.99	0.091	2	0	4	0	123.8	0.154	393	2.75	0	4	6	0	804.6	143
diff	7	50	0	1	3	1	513.0	0.795	1.08	0.030	0	1	3	1	513.0	0.795	38	1.06	0	2	2	1	645.6	36
	9	100	0	1	3	0	242.3	0.529	2.13	0.061	0	1	3	0	242.3	0.529	52	1.49	0	1	1	1	458.2	35
	11	250	2	0	2	0	69.0	0.269	5.65	0.138	2	0	3	0	96.4	0.376	54	1.32	2	1	3	0	256.5	41
mpeg	36	5	1	2	1	1	625.3	0.698	0.61	0.002	1	2	1	1	625.3	0.698	31800	91.12^{+}	1	2	1	2	896.0	349
	42	50	2	2	1	0	361.7	0.554	6.09	0.019	2	2	2	0	382.0	0.585	57400	181.1^{+}	1	2	2	1	652.7	317
	50	200	1	1	1	0	194.6	1.000	27.95	0.070	2	1	1	0	201.7	1.037	209800	521.9^{+}	1	1	1	0	194.6	402
fft	24	200	6	3	6	0	687.2	0.233	262.40	0.020	6	4	5	2	1361.3	0.462	5992	0.453	0	14	6	2	2946.5	13228
	30	100	9	1	5	0	361.0	0.157	134.81	0.011	9	4	8	1	1194.1	0.520	7458	0.584	6	9	10	2	2298.5	12780
	36	100	6	1	4	0	312.3	0.269	90.30	0.006	10	2	9	0	637.8	0.549	12083	0.831	9	6	5	0	1161.3	14534
rand0	20	100	7	2	4	1	750.2	0.491	40.66	0.019	11	2	5	1	806.0	0.527	3125	1.842	0	6	1	2	1529.1	2109
	25	200	7	1	4	0	319.4	0.259	98.56	0.044	9	1	3	1	577.0	0.467	4014	1.802	0	5	6	1	1235.4	2228
	30	200	6	0	3	0	124.9	0.111	116.76	0.042	7	0	4	0	159.4	0.142	4841	1.761	0	5	2	1	1125.8	2749
rand1	23	200	12	3	4	0	675.1	0.307	211.98	0.020	8	6	8	0	1236.4	0.563	7800	0.732	0	10	2	2	2196.7	10659
	28	50	11	1	4	0	347.9	0.178	65.56	0.006	15	3	6	0	751.2	0.385	11219	1.059	0	10	3	1	1953.4	10598
	33	50	9	1	3	0	306.2	0.198	83.26	0.007	16	1	4	0	383.4	0.247	14185	1.235	0	9	4	0	1550.1	11486
rand2	36	10	40	3	13	1	1391.5	0.158	170.24	0.001	51	7	20	1	2301.7	0.262	432211	0.481	0	27	5	16	8789.7	898483
	45	50	32	1	10	0	661.6	0.102	693.58	0.001	42	3	15	2	1731.2	0.268	564703	0.623	0	25	11	8	6468.3	906971
	54	50	26	0	8	0	404.1	0.106	770.99	0.001	43	0	13	0	661.9	0.173	635786	0.614	0	20	13	1	3827.9	1035776
AVG1 (compare to MWIS)								**0.382**		**0.038**						**0.480**		**2.004**					**1.000**	**1.000**
AVG2 (compare to BCGS)								**0.796**		**0.019**						**1.000**		**1.000**					**2.083**	**0.499**

Table 2.7: Scheduling combined with FU and register binding for leakage power minimization

DFG	T_{con}	Scheduling w/o FU and REG Binding										Scheduling with FU and REG Binding										[23] w/o Binding
		ADD		MUL		# of	Int.# of	Leakage Power(W)				ADD		MUL		# of	Int.# of	Leakage Power(W)				
		hvt	*lvt*	*hvt*	*lvt*	REGs	MUX	P_{FU}	P_{REG}	P_{MUX}	P_{total}	*hvt*	*lvt*	*hvt*	*lvt*	REGs	MUX	P_{FU}	P_{REG}	P_{MUX}	P_{total}	P_{total}
ad2	47	0	2	0	3	25	143	1132.2	162.5	177.8	1472.5	0	2	0	3	23	121	1132.2	149.5	155.3	1437.0 (0.976)	1472.5 (1.000)
	70	0	1	3	0	25	131	242.3	162.5	167.7	572.5	0	1	3	0	25	124	242.3	162.5	162.4	567.2 (0.991)	788.4 (1.377)
	94	1	0	2	0	25	114	61.9	162.5	146.7	371.1	1	0	2	0	23	105	61.9	149.5	137.4	348.8 (0.940)	377.6 (1.018)
ae	72	0	1	1	1	34	154	458.2	221.0	169.8	849.0	1	1	1	1	31	146	465.3	201.5	157.8	824.6 (0.971)	770.9 (0.900)
	90	1	1	3	0	36	170	249.4	234.0	196.3	679.7	2	1	3	0	34	157	256.5	221.0	187.1	664.6 (0.978)	684.8 (1.008)
	108	1	0	2	0	36	150	61.9	234.0	176.8	472.7	1	0	2	0	31	149	61.9	201.5	176.3	439.7 (0.930)	493.4 (1.044)
ar	13	0	2	4	2	6	40	971.1	39.0	58.0	1068.1	1	2	4	2	5	35	978.2	32.5	55.1	1065.8 (0.998)	1499.9 (1.404)
	16	0	2	4	0	8	42	429.7	52.0	65.0	546.7	1	2	4	0	6	40	436.8	39.0	61.6	537.4 (0.983)	574.1 (1.050)
	19	2	0	4	0	8	40	123.8	52.0	60.5	236.3	2	0	4	0	6	37	123.8	39.0	56.7	219.5 (0.929)	237.5 (1.005)
diff	7	0	1	3	1	4	15	513.0	26.0	28.6	567.6	0	1	3	1	4	15	513.0	26.0	27.4	566.4 (0.998)	568.7 (1.002)
	9	0	1	3	0	3	15	242.3	19.5	27.3	289.1	0	1	3	0	3	13	242.3	19.5	25.0	286.8 (0.992)	289.1 (1.000)
	11	2	0	2	0	3	11	69.0	19.5	23.8	112.3	2	0	2	0	3	9	69.0	19.5	21.4	109.9 (0.979)	139.7 (1.244)
mpeg	36	1	2	1	1	19	104	625.3	123.5	121.9	870.7	1	2	1	1	17	98	625.3	110.5	111.9	847.7 (0.974)	885.5 (1.017)
	42	2	2	1	0	20	123	361.7	130.0	148.6	640.3	2	2	1	1	19	99	389.1	123.5	125.8	638.4 (0.997)	654.1 (1.022)
	50	1	1	1	0	20	111	194.6	130.0	141.9	466.5	2	1	1	0	19	101	208.8	123.5	117.7	450.0 (0.965)	480.1 (1.029)
fft	24	6	3	6	0	41	358	687.2	266.5	480.9	1434.6	6	3	6	0	39	350	687.2	253.5	474.8	1415.5 (0.987)	2115.2 (1.474)
	30	9	1	5	0	46	340	361.0	299.0	465.4	1125.4	9	1	5	0	45	323	361.0	292.5	462.4	1115.9 (0.992)	1965.0 (1.746)
	36	6	1	4	0	46	346	312.3	299.0	437.0	1048.3	7	1	6	0	41	314	374.2	266.5	398.8	1039.5 (0.992)	1367.3 (1.304)
rand0	20	7	2	4	1	19	168	750.2	123.5	196.6	1070.3	8	2	4	1	17	162	757.3	110.5	183.1	1050.9 (0.982)	1132.6 (1.058)
	25	7	1	4	0	17	159	319.4	110.5	168.1	598.0	8	1	4	0	16	151	326.5	104.0	156.9	587.4 (0.982)	862.1 (1.442)
	30	6	0	3	0	24	157	124.9	156.0	169.5	450.4	6	0	3	0	22	153	124.9	143.0	168.1	436.0 (0.968)	959.9 (1.062)
rand1	23	12	3	4	0	31	289	675.1	201.5	336.4	1213.0	12	3	4	0	28	280	675.1	182.0	328.1	1185.2 (0.977)	1774.3 (1.463)
	28	11	1	4	0	32	291	347.9	208.0	334.6	890.5	13	1	4	0	26	277	355.0	169.0	315.3	846.3 (0.950)	1280.8 (1.438)
	33	9	1	3	0	35	300	306.2	227.5	342.5	876.2	10	1	3	0	34	295	306.2	221.0	310.1	844.5 (0.964)	959.9 (1.096)
rand2	36	40	3	13	1	135	1398	1391.4	877.5	1514.7	3783.6	42	3	15	1	105	1382	1433.1	682.5	1449.3	3564.9 (0.942)	4706.9 (1.244)
	45	32	1	10	0	163	1393	661.6	1059.5	1531.0	3252.1	33	1	11	0	121	1362	696.1	786.5	1426.7	2909.3 (0.895)	4302.2 (1.323)
	54	26	0	8	0	175	1357	404.1	1137.5	1498.3	3039.9	27	0	9	0	124	1310	438.6	806.0	1369.8	2614.4 (0.860)	3304.2 (1.078)
AVG								**1.000**	**1.000**	**1.000**	**1.000**							**1.025**	**0.895**	**0.934**	**0.966**	**1.180**

to larger delays, and potentially the number of FUs will increase, since the sharing ratio among FUs of larger delays is lower than the ratio among FUs of smaller delays.

2.9.4 Interconnection-Aware FU and Register Binding

In this section the impact of binding on leakage power is shown, w

, Int. # of MUXes shows the total number of inputs of all MUXes. P_{FU}, P_{REG} and P_{MUX} are the total leakage powers of FUs, registers and MUXes, respectively; P_{total} is the summation of the three, the overall leakage power. The scheduling w/o FU and REG binding columns show the results that binding is performed only once after scheduling; the scheduling with FU and REG binding show the results with binding being combined into scheduling. The overall leakage power, including register and MUX power, of previous work[23] are also shown in the last column. The power values are obtained by performing binding once on the final scheduling results of[23].

Firstly, the overall leakage power of[23] is 18% larger than mine on average, when binding is performed once after scheduling of both approaches, mainly because less FUs are used in my scheduling solutions. Secondly, in my scheduling solutions with binding, the numbers of FUs slightly increase compared to those without binding, as shown in the columns of ADD and MUL. This is because, w , the objective is to minimize the total power including registers and MUXes, and the solutions with more FUs may have less registers and smaller MUXes. For example, for cases like *ae*, *ar*, *mpeg*, *fft*, *rand*1 and *rand*2, the numbers of FUs increase by 1 or 2, and correspondingly the leakage power of FUs, P_{FU}, increases by 2.5% on average; on the other hand, the numbers of registers and total inputs of MUXes greatly reduce for these cases, thus the overall power also reduces. On average, with binding, the register power reduces by 10.5%, the MUX power reduces by 7.6%, and the overall power reduces by 3.4%, compared to the results without binding. It strongly implies that binding shall be conducted together with scheduling.

2.10 Summary

In this work, a unified low-power scheduling algorithm with multiple threshold or supply voltage technologies is proposed. Two problems are first solved by the unified scheduling algorithm: (1) dynamic power and resource usage co-optimization, and (2) leakage power optimization; besides, the Pareto solutions are discussed for problem (3), dynamic and leakage power co-optimization. The problems are first formulated by ILP, and then are relaxed to LP formulations because the constraint matrices of ILP are totally unimodular. Based on the LP formulation, a mobility graph is built, on which each vertex is associated with a vertex potential. On the mobility graph, the network simplex method is applied to solve the mobility allocation problem, w

, which conducts binding after scheduling only when the scheduling results are promising in reducing current overall power consumption. Experimental results show that, for dynamic power and resource co-optimization the proposed scheduling approach produce optimum solutions for all benchmarks, and for leakage power the scheduling approach excels the latest existing work by 20% leakage power reduction with 52 times speedup. Besides, the combined scheduling and binding results show that integrating binding into scheduling is benefit in reducing overall leakage power consumption.

3

Interconnection Allocation Between Functional Units and Registers in HLS

In this chapter, the second topic, interconnection allocation between FUs and registers, is discussed, w

, called port assignment problem. The purpose is to propose algorithms that can produce optimum solutions with a probability higher than 99%. Besides, the high efficiency is also expected because the port assignment problem is to be solved repeatedly in the interconnection allocation step. The problem is first formulated by ILP, and then solved on a constraint graph. To speedup, a matrix formulation is also to be proposed with pivoting operations of LP. The detailed formulations and algorithms are discussed in the following of this chapter.

3.1 Background and Contributions

In High-Level Synthesis (HLS), interconnections have become one of the key features that influences the performance of integration designs. As shown in[37], interconnections consume a considerable fraction of total circuit power, and[38] shows that interconnections count for more than 50% of dynamic power of Intel microprocessors. Meanwhile, a multiplexer (MUX) is usually expensive in terms of both area and power consumption, especially the ones with large numbers of input ports. As pointed out in[39], the area and power consumption of a 32-to-1 MUX are almost equivalent to a 18-bit multiplier. Therefore, it is important to reduce the complexity of the interconnections that stitch the functional units (FU) and registers, as well as MUX power and area.

The port assignment is an important step in connectivity binding in the High-Level Synthesis to reduce interconnection complexity. Given a fixed binding of FUs and registers, the interconnection is allocated from the registers to FUs through MUXes, w

's work[51] and is proved to be NP-Complete. When the operators are all

binary commutative, it is more accurately defined as the Port Assignment Problem for Binary Commutative Operators (PAP-BCO) in[55]. Since the port assignment is performed on each FU after binding, its high quality solutions are highly expected since they directly determine the interconnection complexity and MUX usage. Besides, if the port assignment can be combined into register binding step, it can help evaluate the solution quality of register binding in terms of interconnection complexity. Therefore, the execution time of port assignment algorithm must be short, w , for example[27], and being solved repeatedly. Consequently, producing optimal solutions, and estimating interconnection complexity quickly and accurately, are the essential demands for port assignment algorithms.

There are already several literatures that deal with the port assignment problem. The work in[52] performed global permutation of all the inputs of a FU during MUX generation, and the work in[53] designed an integer linear programming algorithm; for both algorithms the complexity is a concern. Chen and Cong in[54] proposed a greedy algorithm by swapping the operands of an operator if some operands are assigned to the registers that drive both ports. However it has a limitation that operand swapping will not help when there is a series of operations that have circular dependencies. The work in[55] tried to decrease the MUX size discrepancy between the two ports. It reformulated the PAP-BCO problem as PAP-BCO*, which accounted for evenly distributing input signals among two multiplexers, but it did not propose practical algorithms for PAP-BCO problem.

I also have published several works for port assignment problem[33,34,36]. [33] was the first to propose a practical spanning tree based algorithm, where elementary tree transformation is adopted for solution optimization. [34] extended the algorithm from[34] to also take MUX power into consideration. [36] still adopted the spanning tree for initial solution generation, but used a Fiduccia and Mattheyses (FM) based method for optimization instead of elementary tree transformation. All these proposed algorithms excel other existing work say[54], but they still have problems. For example in[33] and[34], the elementary tree transformation used for solution optimization was time consuming, which degraded the algorithm efficiency. In[36], for port assignment solution perturbation, the FM based algorithm only allowed one vertex being moved or two vertices being swapped in each iteration of local search; thus the search area was small, which limited the solution optimality.

Therefore in this work, I improve my earlier works[33] and[34] by proposing a new matrix based algorithm, and compare it to[36] and another existing work[54]. The contributions are stated as the following, among which (1) and (2) have been proposed in[33] and[34], and (3) to (5) are additional contributions in this paper:

1. A **spanning tree and conflict graph** based method is first proposed to obtain feasible solutions for port assignment problem. It not only generates high quality initial solutions, but can also estimate interconnection complexity during register binding with little time cost.

2. An **elementary tree transformation** based method is proposed for solution optimization. In each iteration, the spanning tree structure is changed by elementary tree transformation, to get a smaller conflict graph.

3. To improve the efficiency of solution optimization, a **matrix formulation** is proposed by defining two operations $\oplus$ and $\otimes$; because the coefficient matrix agrees with the **Simplex Tableau** format, **Simplex Method** is adopted and **pivotings** are performed for optimization.

4. To improve the efficiency of **pivot selection**, the properties of pivotings are studied and **successive pivotings** are proposed; three pivoting rules are proposed to filter unpromising or redundant pivoting calculations, which significantly speeds up the pivoting selection.

5. My algorithm is tested on real benchmarks to evaluate the improvements in terms of power, area and delay; moreover, the testbench FFT is implemented on FPGA to get actual improvements of power, area and clock frequency achieved by my algorithm.

The rest of this paper is organized as follows. Section 4.2 describes the problem formulation, Section 3.3 introduces the initial solution generation. Section 3.4 illustrates the elementary tree transformation based solution optimization. In Section 3.5 the matrix formulation is proposed and Simplex Method is applied for optimization speedup. Section 3.6 shows the experimental results and followed by conclusion and future work in Section 3.7.

3.2 Problem Formulation

After scheduling and FU and register binding in high-level synthesis, the final step is to connect allocated registers to FUs through MUXes. For each allocated FU, denoted as *fu*, a number of operations $\{op_1, op_2, \ldots, op_k\}$ are carried on it sequentially. The input operands of op_1 to op_k are stored in the allocated registers, w 's input ports through MUXes. The step is called *port assignment*, and it is performed on one FU each time.

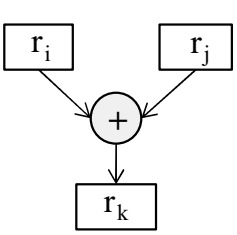
When the *fu* is a binary commutative operator with two input ports, say "+" or "×", the problem is called the *Port Assignment Problem for Binary Commutative Operators (PAP-BCO)*. In this work only the binary commutative FUs are considered. Therefore, each *fu* has two ports, and a register can be connected to *fu*'s left or right port, or to both ports.

The objective of port assignment is to minimize the number of interconnections that connect the two ports of a FU to its input registers through MUXes. Naturally, the widths of two MUXes are also minimized.

Fig.3.1(a) shows an example of a FU that carries "+" operation with two input ports, and Fig.3.1(b) shows the sequential operations that are executed on it. In control step c_{t_1}, the operands stored in registers r_1 and r_2 are executed on the FU, and in c_{t_2}, the operands stored in r_1 and r_3 are executed. Fig.3.1(c) shows the port assignment result for this example. In this case the number of interconnections between the registers and the FU ports is 6, which is the objective to be minimized.

Besides, since wider MUXes have larger signal delays that may affect the critical path delay of a circuit, the size of the largest MUX in a circuit is also minimized as a secondary objective in this work.

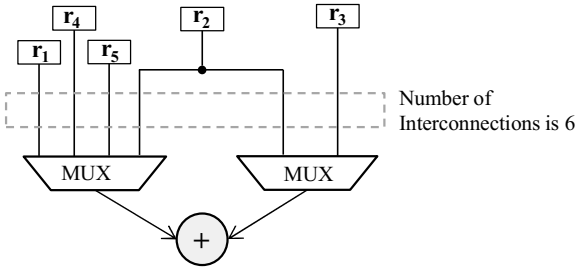
In the remaining of this section, first the graph based formulation is introduced, and followed by an ILP formulation.



(a) An example of a function unit that carries "+" operation

c_{t_1}: $r_1 + r_2$
c_{t_2}: $r_1 + r_3$
c_{t_3}: $r_3 + r_4$
c_{t_4}: $r_5 + r_3$
c_{t_5}: $r_2 + r_3$
c_{t_6}: $r_2 + r_3$

(b) The operations executed in sequential control steps



(c) Interconnections allocated between registers and *fu* ports

Figure 3.1: An example of a binary commutative functional unit with the interconnection allocation between the registers and MUXes.

3.2.1 Graph-Based Formulation

In this work, the port assignment problem is formulated on an undirected simple graph $G = (V, E)$, called the *constraint graph*. $V = \{v_1, v_2, \ldots, v_m\}$ is the set of registers that store input operands of *fu*, and are to be connected to *fu*; $E = \{e_1, e_2, \ldots, e_n\}$, where each $e = (v_i, v_j) \in E$ means registers r_i and r_j are providing two operands for *fu* in one operation.

To indicate the port which a register is connected to, vertex set V is denoted as $V = V_L \cup V_R \cup V_B$ ($V_L \cap V_R = \varnothing$, $V_L \cap V_B = \varnothing$, $V_R \cap V_B = \varnothing$); a vertex $v_i \in V_L$ if its corresponding register r_i is connected to *fu*'s left port, $v_i \in V_R$ if r_i is connected to *fu*'s right port, or $v_i \in V_B$ if r_i is connected to both ports. Therefore, the port assignment problem is also regarded as a *vertex partition problem*, and hereafter the vertex partition and port assignment are used indiscriminately.

For each edge $e = (v_i, v_j)$, their corresponding registers r_i and r_j shall be connected to different ports of *fu* separately, or at least one of them is connected to both ports, like r_2 shown in Fig.3.1(c), to guarantee that the operands stored in r_i and r_j can be provided to *fu* in the same control step. Therefore, the vertex partition is valid if and only if when each edge $e = (v_i, v_j)$ satisfies the following *Partition Rules*:

$$v_i \in V_L, v_j \in V_R \tag{3.1}$$

$$v_i \in V_R, v_j \in V_L \tag{3.2}$$

$$v_i \in V_B \text{ or } v_j \in V_B \tag{3.3}$$

As proved in[55], minimizing the interconnections between the *fu* and MUXes equals minimizing the number of registers that are connected to both ports, i.e., the value of $|V_B|$. For example in Fig.3.2(a) and Fig.3.2(c), two different vertex partitions are shown, w

, and in Fig.3.2(d) there are two registers r_1 and r_5 being connected to both ports. The number of registers being connected to both ports are to be minimized.

Therefore, the port assignment problem is formulated as:

Given: An undirected simple graph $G = (V, E)$, where $v_i \in V$ stands for register r_i which is to be connected to functional unit *fu*, and $e = (v_i, v_j) \in E$ stands for an execution of r_i and r_j in a particular control step.

Goal: To find a vertex partition on constraint graph G, where each $v \in V$ is partitioned to V_L, V_R or V_B following the partition rules, to minimize $|V_B|$; after that, $max\{|V_L|, |V_R|\}$ is minimized as a secondary objective.

3.2.2 ILP Formulation

For each vertex $v_i \in V$, three 0-1 variables δ_l^i, δ_r^i, δ_b^i are defined to represents its partition, for example, if $v_i \in V_L$, then $\delta_1^i = 1$, otherwise $\delta_1^i = 0$. They are stated as:

$$\delta_l^i = 1 \; if \; v_i \in V_L; \quad \delta_l^i = 0 \; if \; v_i \notin V_L \tag{3.4}$$

$$\delta_l^i = 1 \; if \; v_i \in V_R; \quad \delta_r^i = 0 \; if \; v_i \notin V_R \tag{3.5}$$

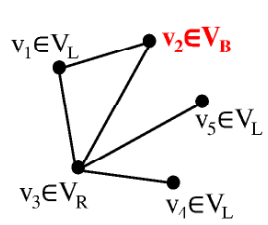
$$\delta_b^i = 1 \; if \; v_i \in V_B; \quad \delta_b^i = 0 \; if \; v_i \notin V_B \tag{3.6}$$

Therefore, the ILP formulation is written as:

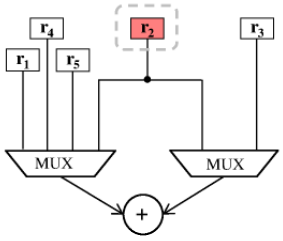
$$\begin{aligned} Min: & \sum_{v_i \in V} \delta_b^i \\ s.t. \; & for \; \forall v_i \in V : \delta_l^i, \delta_r^i, \delta_b^i \in \{0, 1\} \\ & for \; \forall v_i \in V : \delta_l^i + \delta_r^i + \delta_b^i = 1 \\ & for \; \forall e = (v_i, v_j) \in E : \delta_l^i + \delta_l^j \leq 1, \delta_r^i + \delta_r^j \leq 1 \end{aligned} \tag{3.7}$$

This ILP formulation can be solved using a general linear programming tool *lp_solver*[56].

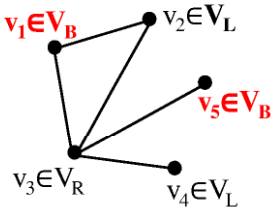
In the following, first, an efficient algorithm is proposed to **generate feasible initial solutions** for port assignment problem in Section 3.3. Given an initial solution, an **elementary tree transformation based optimization** is proposed to improve the solution quality, discussed in Section 3.4. To improve the efficiency of tree transformation based optimization, another optimization method, **the Simplex Method based optimization**, is proposed in Section 3.5. The properties of pivotings and successive pivotings are studied as well to speedup the Simplex Method based optimization.



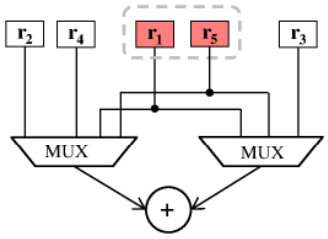
(a) One valid vertex partition



(b) Port assignment from (a) with 6 interconnections



(c) Another valid vertex partition



(d) Port assignment from (c) with 7 interconnections

Figure 3.2: The examples of vertex partitions and the corresponding port assignment solutions.

3.3 Conflict Graph Based Initial Solution Generation

In this section, a heuristic method is proposed to find a valid initial port assignment solution. First, vertices are initially partitioned by a spanning tree, which is introduced in Section 3.3.1, and then the vertex partition is legalized by a **minimum vertex cover** based method on a **conflict graph**, which is introduced in Section 3.3.2.

3.3.1 Initial Vertex Partition on the Spanning Tree

Some graph notations from[58] that will be used in this paper are given:

- A *spanning tree* of graph G is denoted as T. The *tree edges* on the spanning tree are denoted as e_t, and the *non-tree edge* are denoted as e_c.
- A *fundamental cycle*, also called *fundamental loop*, is denoted as FL. Each fundamental loop FL has only one non-tree edge e_c, and each e_c has a unique fundamental loop, denoted as $FL(e_c)$.
- A *fundamental cutset* is denoted as FC. Similarly, each tree edge e_t has a unique fundamental cutset, denoted as $FC(e_t)$.
- The fundamental loop size refers to the number of edges in the loop. The *parity* of a non-tree edge e_c, denoted as $p(e_c)$, is defined on the size of fundamental loop $FL(e_c)$: if the loop size is odd, the parity of e_c is *odd*, and e_c is an *odd edge*; otherwise the parity of e_c is *even* and e_c is an *even edge*.

On the given constraint graph $G = (V, E)$, a spanning tree T_o rooted v_r is first built, and E is written as $E = E_e \cup E_o \cup E_e$, where E_t is the set of tree edges, E_o is the set of odd non-tree edges, and

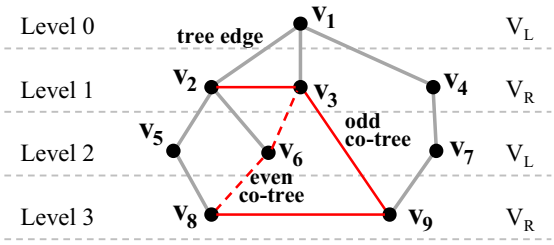
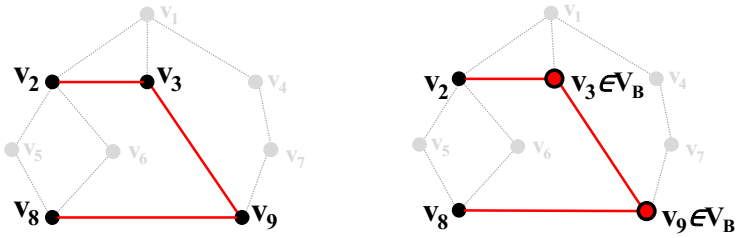


Figure 3.3: Example of tree edges, odd non-t -tree edges.



(a) The conflict graph

(b) The vertex cover VC on the conflict graph

Figure 3.4: The example of initial vertex partition legalization.

E_e is the set of even non-tree edges. The *vertex level*, denoted as $lvl(v)$, is defined as the distance from vertex v to root v_r on the tree. Then the initial vertex partition is applied according to the vertex levels as:

$$v \to V_L \; if \; lvl(v) \; is \; even \tag{3.8}$$

$$v \to V_R \; if \; lvl(v) \; is \; odd \tag{3.9}$$

Fig.3.3 shows an example of initial vertex partition on a spanning tree. The gray edges are tree edges; the level of vertex v_1 is 0, the level of vertices v_2, v_3, and v_4 is 1, etc. The vertices whose levels are 0 and 2 are initially partitioned to V_L, 1 , v_5, v_6 and v_7, and the vertices whose levels are 1 and 3 are partitioned to V_R.

Given the initial partition, there is a theorem as the following:

Theorem 1. Each odd non-tree edge violates the partition rules, while tree edges and even non-tree edges do not.

The proof is straightforward and is omitted. For example in Fig.3.3, the odd non-tree edge (v_3, v_9) violates the partition rules because both v_3 and v_9 are partitioned to set V_R, while the even non-tree edge (v_6, v_8) does not.

3.3.2 Solution Legalization: Vertex Cover on Conflict Graph

The initial vertex partition is usually non-valid because odd non-tree edges violate the partition rules; these edges are defined as **conflict edges**. To record all conflict edges, a conflict graph composed of

conflict edges is defined as:

Definition 1. A graph $G_c = (V_c, E_c)$ is defined as the **conflict graph**, where $e \in E_c$ is the set of conflict edges, and $v \in V_c$ are the endpoints of conflict edges.

Fig.3.4(a) shows the conflict graph built from Fig.3.3, where the edges (v_2, v_3), (v_3, v_9) and (v_8, v_9) are conflict edges.

To legalize the initial vertex partition, some vertices on the conflict graph are needed to be repartitioned to to V_B, to make all conflict edges satisfy the partition rules. For each conflict edge $e_c = (v_i, v_j)$, if either v_i or v_j or both of them are re-partitioned to V_B, edge e_c is legalized according to partition rule 3. On the other hand, since the objective of vertex partition is to minimize $|V_B|$, the number of vertices that are repartitioned to V_B is expected to be as small as possible.

Consequently, a **minimum vertex cover** is proposed to determine the vertices to be repartitioned on the conflict graph:

Definition 2. A **vertex cover** of a graph G is a set of vertices that each edge of the graph is adjacent to at least one vertex in the set. A **minimum vertex cover**, denoted as VC, is a vertex cover with the smallest number of vertices in it.

Theorem 2. Given a conflict graph G_c and its minimum vertex cover VC, the initial vertex partition is legalized by repartitioning all the vertices $v \in VC$ to V_B.

Proof. According to the definition of a vertex cover, each edge $e = (v_i, v_j)$ of G_c is adjacent to at least one vertex $v \in VC$, and $v \in V_B$; therefore for each $e = (v_i, v_j)$, either v_i or v_j is in V_B, which no longer violates the partition rule. □

Though finding a minimum vertex cover is NP-hard, many high-quality algorithms have been proposed these years. Moreover, if minimum vertex cover is to be found on a graph which is a tree, it is reduced to a class-P problem and optimal solutions can be easily obtained. For port assignment problem, from pre-experiments, it shows that the conflict graphs are mostly sparse graphs and almost 90% of them are trees. Therefor, in most cases optimal minimum vertex cover can be obtained efficiently. In this work a simple greedy heuristic in[57] is adopted to solve minimum vertex cover problem.

Fig.3.4 shows an example of vertex partition legalization. Fig.3.4(a) shows the conflict graph built from Fig.3.3, and Fig.3.4(b) shows its minimum vertex cover including v_3 and v_9. By repartitioning v_3 and v_9 to V_B, the vertex partition is legalized.

In this section the algorithm to produce a valid port assignment solution is proposed. As discussed in the introduction, this is practical for interconnection complexity estimation during the register binding because of its high efficiency. Moreover, as to be shown in the experiments, the initial solutions have only 20% overhead compared to optimum solutions by ILP, which can be regarded as a good approximation for interconnections.

Given the NP-completeness of the port assignment problem, however, further optimizations are still needed to get near-optimal or optimal solutions. My proposed solution optimizations are introduced in the following sections.

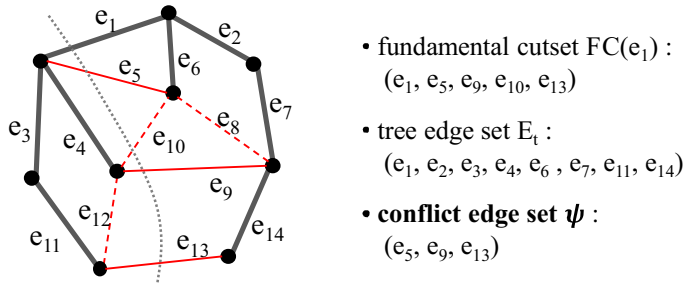


Figure 3.5: Example of fundamental cutset *FC*, tree edge set E_t and conflict edge set ψ.

3.4 Elementary Tree Transformation Based Solution Optimization

In this section the elementary tree transformation based optimization is introduced. Firstly some definitions and the objective of solution optimization is given in Section 3.4.1. Then an **elementary tree transformation** based local search method is discussed in Section 3.4.2 and Section 3.4.3. Also, two important properties of elementary tree transformation to speed up the optimization are given.

3.4.1 Solution Optimization

Given a specific spanning tree T on the constraint graph, a unique conflict graph G_c is determined; then the minimum vertex cover is found on G_c and a valid vertex partition solution is obtained. Since most of the conflict graphs are trees or near-trees, the chance to get an optimal minimum vertex cover is high, so that the vertex partition solution is predominantly determined by spanning tree. Therefore, changing spanning tree structure greatly affects solution quality.

On the other hand, calculating the minimum vertex cover every time is time consuming. Since empirically the minimum vertex cover size is positively correlated to the number of conflict edges, the objective of solution optimization is to minimize the number of conflict edges for efficiency, defining:

Definition 3: A **conflict edge set**, denoted as ψ, is the set of all the conflict edges, i.e., the odd non-tree edges. The number of edges in ψ is defined as the size of ψ, denoted as $\|\psi\|$.

Fig.3.5 shows examples of fundamental cutset, tree edge set and conflict edge set. The edges of the spanning tree are bold ones in the set E_t; the fundamental cutset of tree edge e_1 is $FC(e_1)$, including e_1, e_5, e_9, e -tree edges e_5, e_9 and e_{13}, and $\|\psi\|$ is 3.

To sum up, the solution optimization is stated as: by iteratively changing the structure of the spanning tree on the constraint graph, the value of $\|\psi\|$ is minimized.

3.4.2 Elementary Tree Transformation

For solution optimization, a local search based algorithm is proposed; in each iteration, the structure of spanning tree is changed using **elementary tree transformation**. It is one of the most widely-used transformations when computing minimum-cost tree problems as explained in[61], defined as:

Definition 4: The **elementary transformation** of a spanning tree T refers to the process that, a tree edge e_t being removed out of T, resulting in the disjoint of two sub-trees, and an edge $e_c \neq e_t$ being added to into the tree so that the two sub-trees are connected as a new spanning tree T'[57].

Accordingly, an elementary transformation can be uniquely defined by two specific edges, a tree e_t and a non-tree edge in the fundamental cutset of e_t, i.e., $e_c \in FC(e_t)$; thus, an elementary transformation is denoted as $ET(e_t, e_c)$.

Specific to this problem, there are two properties of the elementary tree transformation, which can be used to speedup the local search based optimization by avoiding redundant calculations of $ET(e_t, e_c)$ or $FC(e_t)$.

Property 1: For an elementary transformation $ET(e_t, e_c)$, the conflict edge set ψ is updated as:

$$\psi' = \begin{cases} \psi, & p(e_c) = 0 \\ \psi \triangle FC(e_t), & p(e_c) = 1 \end{cases} \tag{3.10}$$

where $p(e_c)$ is the parity of non-tree edge e_c, and the $\triangle$ is the *symmetric difference* between two sets defined as:

$$A \triangle B = (A \cup B) - (A \cap B) \tag{3.11}$$

Property 2: For an elementary transformation $ET(e_t, e_c)$, the fundamental cutset for each tree edge e_t is updated as:

$$FC'(e_t) = \begin{cases} FC(e_t), & e_t \notin FL(e_c) \\ FC(e_t) \triangle FC(e_t), & e_t \in FL(e_c) \end{cases} \tag{3.12}$$

where $FL(e_c)$ is the fundamental loop of non-tree edge e_c.

Property 1 shows that when performing an elementary tree transformation $ET(e_t, e_c)$, if the non-tree edge e_c is an even edge, the conflict edge set ψ remains unchanged; therefore these transformations shall be avoided. Property 2 shows that after performing an elementary tree transformation, the fundamental cutsets of tree edges can be calculated through set operations, and only a part of the fundamental cutsets are needed to be updated. The proofs are given in Section 3.5 through matrix formulations and operations.

3.4.3 Local Search Based Optimization Algorithm

Adopting the elementary tree transformation as local transformation, a local search based algorithm is proposed shown in Algorithm 1, including both initial solution generation and solution optimization.

First, the outermost cycle is the random start as shown in line 1. In each iteration of the random start, an initial spanning tree is first built on the constraint graph $G = (V, E)$, and initial port assignment solution is obtained, shown in line 2 and 3; and then, local search based solution optimization is performed shown in line 4 to 20.

In each iteration of local search, first α candidate edge pairs for elementary transformation are collected shown in line 5 to 9. For each tree edge e_t, and all the non-tree edges $e_c \in FC(e_t)$, the value of $\|\psi'\|$ is computed for each edge pair (e_t, e_c), but the elementary transformation is not actually performed. Total α candidate edge pairs in set *Cdt* are kept with smallest $\|\psi'\|$.

Then as shown in line 11 to 15, the elementary transformation is performed on each edge pair in

Algorithm 3 Local Search Based Port Assignment Algorithm

Require: Constraint graph $G = (V, E)$
Ensure: Find a valid vertex partitioning of each v

```
1: while stop criteria of random start NOT met do
2:     Build an initial spanning tree T
3:     Calculate the initial port assignment solution
4:     while 1 do
5:         for each tree edge e_t ∈ E_t do
6:             for each e_c ∈ FC(e_t) do
7:                 k ← the value of ||ψ'|| is computed
8:                 If k is small enough, keep (e_t, e_c) in set Cdt
9:             end for
10:        end for
11:        for each candidate pair (e_t, e_c) ∈ Cdt do
12:            Perform transformation ET(e_t, e_c) without updating T
13:            Find minimum vertex cover on the new conflict graph
14:            Keep the best solution as (e_to, e_co)
15:        end for
16:        if current solution not improved then Break
17:        Choose an odd non-tree e_co ∈ FC(e_to) randomly
18:        Apply transformation ET(e_c, e_co), update tree T
19:    end while
20: end while
```

Cdt, and the minimum vertex cover on the conflict graph is found. The best solution (e_{t_o}, e_{c_o}) with the minimum $|V_B|$ is kept. The solution (e_{t_o}, e_{c_o}) is accepted if it improves the current global best solution, otherwise the local search stops.

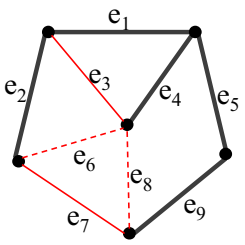
3.4.4 Critical Path Optimization

As described in Section 4.2, critical path delay is the secondary objective function of port assignment problem. Therefore, after $|V_B|$ being minimized, the size of the largest MUX among all MUXes is minimized, because among all control steps, the one with the largest MUX has a much higher possibility to be the critical path.

Critical path optimization can be performed by selecting from a group of port assignment solutions. During the optimization, many solutions may be found which have the same value of $|V_B|$ but different values of V_L and V_R; therefore from the solutions which already have minimized $|V_B|$, the one with minimized $max\{|V_L|, |V_R|\}$ is chosen.

3.5 Simplex Method Based Solution Optimization

In Section 3.4, the elementary tree transformation based solution optimization is introduced. Although the two properties can speed up the algorithm, the tree transformation is still time



Edge Status Vector ψ

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\psi\|$
ψ	0	0	1	0	0	0	1	0	0	2

Fundamental Cut Vector FC

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	
FC(e_1)	1	0	1	0	0	1	1	0	0	
FC(e_2)	0	1	0	0	0	1	1	0	0	
...					...	...				

Figure 3.6: Examples of fundamental cut vector *FC* and edge status vector ψ.

consuming. To further improve the efficiency of solution optimization, in this section, a **new matrix formulation** is first proposed by defining two mathematic operations, $\oplus$ and $\otimes$, which is introduced in Section 3.5.2. Since the coefficient matrix of this proposed formulation meets the **Simplex Tableau format**, the **Simplex Method** is adopted to efficiently perform the optimization, which is introduced in Section 3.5.3. Moreover, to improve both efficiency and optimality, the properties of **pivotings and successive pivotings** specific to port assignment problem are discussed. Based on these properties, a **pivoting selection** method, and a **two-step pivoting selection** method are proposed, which significantly contribute to algorithm speedup and optimality improvement.

3.5.1 Preliminaries and Mathematic Formulation

Fundamental cutset and edge status vector

In Section 3.4 the fundamental cut set *FC*, fundamental loop set *FL* and conflict edge set ψ are defined. In this section, in order to organize the formulation into a matrix, *vectors* are proposed to instead *edge sets*. Without confusion, the same notations of *FC*, *FL* and ψ are still adopted to represent edge vectors.

Definition 5: A **fundamental cutset vector** of tree edge e_t, denoted as $FC(e_t)$, is a 0-1 column vector with an entry for each edge on the constraint graph $G = (V, E)$: if edge e is in the fundamental cut set of e_t, its entry is 1; otherwise is 0.

Definition 6: An **edge status vector**, denoted as ψ, is also a 0-1 column vector with an entry for each edge: if edge e is a tree edge or even non-tree edge, its entry is 0; if e is an odd non-tree edge, i.e., the conflict edge, its entry is 1. The number of 1-entries is the *size* of ψ, denoted as $\|\psi\|$.

Fig.3.6 shows examples of edge status vector ψ, and fundamental cutset vectors $FC(e_1)$ and $FC(e_2)$.

Mathematic Formulation

To formulate the port assignment optimization mathematically, an exclusive-or operation is first defined.

Definition 7: The $\oplus$ operation between two 0-1 values is defined as:

$$x \oplus y = \begin{cases} 1, & x = 1, y = 0 \ or \ x = 0, y = 1; \\ 0, & x = 0, y = 0 \ or \ x = 1, y = 1. \end{cases} \tag{3.13}$$

Given an initial spanning tree and initial vertex partition on the constraint graph $G = (V, E)$, where

each $v \in V$ has been assigned to V_L or V_R, the variables v and e are defined as:

$$v = \begin{cases} 0, & v \in V_L \\ 1, & v \in V_R \end{cases} \tag{3.14}$$

$$e = \begin{cases} 0, & e \text{ is tree edge or even non-tree edge} \\ 1, & e \text{ is odd non-tree edge (conflict edge)} \end{cases} \tag{3.15}$$

The definition of e is consistent with the parity of non-tree edges, and the parity of tree edges are regarded as 0. Also, it is consistent with the definition of the edge status vector ψ.

For each conflict edge where $e = 1$, it has:

$$for\ each\ e_k = (v_i, v_j) \in E : v_i \oplus v_j \oplus e_k = 1 \tag{3.16}$$

If $\oplus$ operations are performed among the formulas in Eq.3.16, where all edges exactly form a *loop* in the graph G, the variables v_i can be completely eliminated because $v_i \oplus v_i = 0$. Therefore, for each fundamental loop $FL(e_c)$, it has:

$$\begin{gathered} for\ each\ FL(e_c) = \{e_{i_1}, e_{i_2} \ldots, e_{i_k}\} : \\ e_{i_1} \oplus e_{i_2} \oplus \ldots \oplus e_{i_k} = p(e_c) \end{gathered} \tag{3.17}$$

Eq.3.17 shows the mathematic constraints of port assignment optimization; the objective is to minimize $\|\psi\|$. Fig.3.7(a) shows an example. For fundamental loop $FL(e_3) = \{e_1, e_3, e_4\}$, it has a constraint of $e_1 \oplus e_3 \oplus e_4 = 1 = p(e_3)$, where the loop size of $FC(e_3)$ is 3 and the parity of e_3 is $p(e_3) = 1$.

3.5.2 Matrix Formulation

Definition 8: The $\oplus$ operation between two matrices $\mathcal{A}$ and B, where $\mathcal{A}$ and B are both $m \times n$ with only 0-1 entries, is defined as:

$$(\mathcal{A} \oplus B)_{ij} = \mathcal{A}_{ij} \oplus B_{ij} \tag{3.18}$$

Definition 9: The $\otimes$ operation between two matrices $\mathcal{A}$ and B, where $\mathcal{A}$ is a $n \times m$ matrix and B is a $m \times p$ matrix with 0-1 entries, is defined as the ordered multiplication and exclusive-or operations as the following:

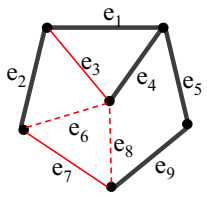
$$(\mathcal{A} \otimes B)_{ij} = (\mathcal{A}_{i1} \cdot B_{1j}) \oplus (\mathcal{A}_{i2} \cdot B_{2j}) \oplus \cdots \oplus (\mathcal{A}_{im} \cdot B_{mj}) \tag{3.19}$$

According to the theory of Fundamental Cycle Matrix (FCM) Theory[58], the formulas in Eq.3.17 is written into the following equation:

$$B \otimes E = P \tag{3.20}$$

w ( )

) shows the matrix formula organized from Fig.3.7(a) w



FL(e_3): $e_1 \oplus e_3 \oplus e_4$ = 1 = p(e_3)
FL(e_6): $e_1 \oplus e_2 \oplus e_4 \oplus e_6$ = 0 = p(e_6)
FL(e_8): $e_4 \oplus e_5 \oplus e_8 \oplus e_9$ = 0 = p(e_8)
FL(e_7): $e_1 \oplus e_2 \oplus e_5 \oplus e_7 \oplus e_9$ = 1 = p(e_7)

(a) The tree and non-t .

$$\mathbf{B} \qquad \otimes \ \mathbf{E} \ = \ \mathbf{P}$$

$$\begin{array}{c} e_1\ e_2\ e_3\ e_4\ e_5\ e_6\ e_7\ e_8\ e_9 \\ \begin{bmatrix} 1 & & 1 & 1 & & & & & \\ 1 & 1 & & 1 & & 1 & & & \\ & & & 1 & 1 & & & 1 & 1 \\ 1 & 1 & & & 1 & & 1 & & 1 \end{bmatrix} \end{array} \otimes \begin{bmatrix} e_1 \\ \vdots \\ e_9 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(

$$(\ \mathbf{I} \qquad \mathbf{A}\ ) \qquad \otimes \ \mathbf{E} \ = \ \mathbf{P}$$

$$\begin{array}{c} e_3\ e_6\ e_7\ e_8\ e_1\ e_2\ e_4\ e_5\ e_9 \\ \begin{array}{c} e_3 \\ e_6 \\ e_7 \\ e_8 \end{array} \begin{bmatrix} 1 & & & & 1 & & 1 & & \\ & 1 & & & 1 & 1 & 1 & & \\ & & 1 & & 1 & 1 & & 1 & 1 \\ & & & 1 & & & 1 & 1 & 1 \end{bmatrix} \end{array} \otimes \begin{bmatrix} e_3 \\ \vdots \\ e_8 \\ e_1 \\ \vdots \\ e_9 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

parity of e_7: p(e_7)

fundamental loop of e_7
FL(e_7) = {e_7, e_1, e_2, e_5, e_9}

fundamental cutset of e_1
FC(e_1) = {e_1, e_3, e_6, e_7}

) Write into fundamental cycle matrix format (Simplex Tableau format)

Figure 3.7: The example of mathematic formulation and its matrix formulation.

combination of a $n_c \times n_c$ square identity matrix I and a $n_c \times n_t$ matrix A, written as:

$$B = (I\ A) = (b^1 \cdots b^p \cdots b^{n_c}\ b^{n_c+1} \cdots b^q \cdots b^n) =$$

$$\begin{array}{c} \quad e_{c_1} \quad \cdots \quad \boldsymbol{e_{c_p}} \quad \cdots \quad e_{c_{n_c}} \quad e_{t_1} \quad \cdots \quad \boldsymbol{e_{t_q}} \quad \cdots \quad e_{t_{n_t}} \\ \begin{array}{c} e_{c_1} \\ \vdots \\ \boldsymbol{e_{c_p}} \\ \vdots \\ e_{c_{n_c}} \end{array} \begin{pmatrix} 1 & & & & & b_{11} & & b_{1q} & & b_{1n_t} \\ & \ddots & & & & \vdots & & \vdots & & \vdots \\ & & 1 & & & b_{p1} & & b_{pq} & & b_{pn_t} \\ & & & \ddots & & \vdots & & \vdots & & \vdots \\ & & & & 1 & b_{n_c1} & & b_{n_cq} & & b_{n_cn_t} \end{pmatrix} \end{array} \tag{3.21}$$

w , i.e., e_{c_1} to $e_{c_{n_c}}$; the matrix A represents the coefficients of the tree edge variables, i.e., e_{t_1} to $e_{t_{n_t}}$. The column vector $b^i(n_c+1 \le i \le n)$ represent the fundamental cutset of the tree edge e_i, and the row vector $r^j(1 \le j \le n_c)$

represents the fundamental loop of the non-tree edge e_i.

Letting E_c represent the non-tree edge variables and E_t represent the tree edge variables, E is written as $E = (E_c\ E_t)^T$, and Eq.3.20 is written as:

$$\begin{bmatrix} I & \mathcal{A} \end{bmatrix} \otimes \begin{bmatrix} E_c \\ E_t \end{bmatrix} = P \tag{3.22}$$

Fig.3.7(c) shows the matrix format of Eq.3.22, where the coefficient matrix B is organized as a fundamental cycle matrix.

Therefore, the matrix formulation of port assignment optimization is stated as:

$$\begin{aligned} Minimize: \ \|\psi\| &= \sum_{e_i \in E_c} p(e_i) \\ s.t.: \ B \otimes E &= P \end{aligned} \tag{3.23}$$

3.5.3 Simplex Tableau Format

The formulation shown in Eq.3.23 agrees with the Linear Programming Problem (LPP) standard form because:

1. All variables are non-negative, i.e., all 0-1 variables;
2. The coefficient matrix $B = (I\ \mathcal{A})$ agrees with the Simplex Tableau format, where the matrix I is an identity submatrix, whose variables are *basic variables*, and the variables in $\mathcal{A}$ are *nonbasic variables*.
3. When all nonbasic variables are 0, the right-hand value are determined by basic variables: in this formulation the nonbasic variables are tree edges and the basic variables are non-tree edges; the edge parities, i.e., the right-hand values, are determined by the $\oplus$ operations of all non-tree variables.

The differences between the proposed formulation and LPP are:

1. The proposed objective function is not a linear combination of all variables while the LPP is;
2. The proposed formulation is composed of $\oplus$, $\otimes$ and multiplications, while LPP is composed of additions, substractions and multiplications.

Therefore the proposed formulation is not guaranteed to get optimum solutions as the linear programming using Simplex Method. However, since it agrees the LPP form, the similar **pivoting** of Simplex Method can be adopted.

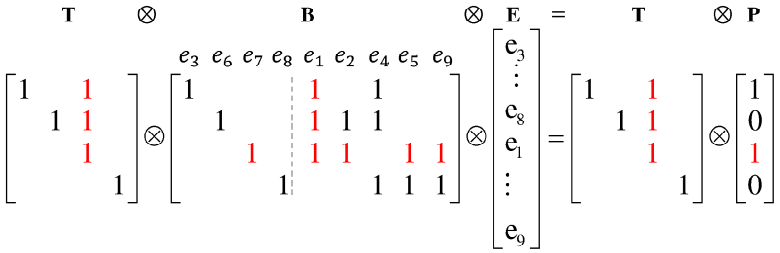
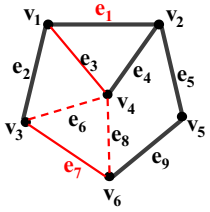
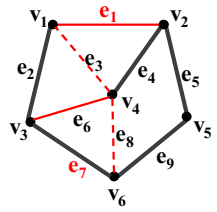


Figure 3.8: Example of the pivoting using pivoting matrix T.



Original edge status vector and fundamental cutset vectors

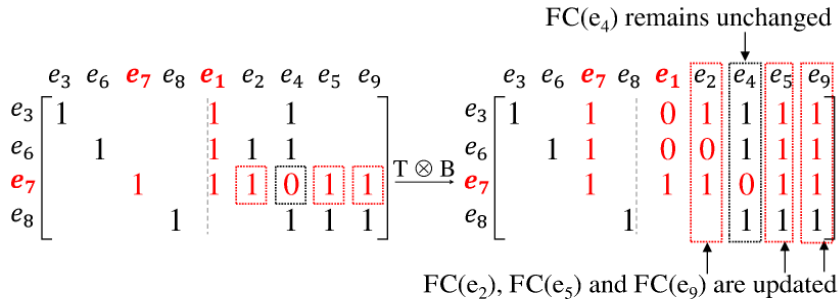
	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
ψ	0	0	1	0	0	0	1	0	0
$FC(e_1)$	1		1			1	1		
$FC(e_2)$		1				1	1		
$FC(e_4)$			1	1		1		1	
$FC(e_5)$					1		1	1	
$FC(e_9)$							1	1	1



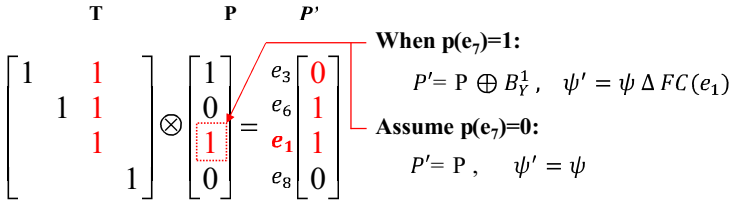
Edge status vector and fundamental cutset vectors after **ET(e_1, e_7)**

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	
ψ'	1	0	0	0	0	1	0	0	0	
$FC'(e_7)$	1		1			1	1			
$FC'(e_2)$	1	1	1							$FC'(e_2) = FC(e_2)\,\Delta\,FC(e_1)$
$FC'(e_4)$			1	1		1		1		$FC'(e_4) = FC(e_4)$
$FC'(e_5)$	1		1		1	1		1		$FC'(e_5) = FC(e_5)\,\Delta\,FC(e_1)$
$FC'(e_9)$	1		1			1		1	1	$FC'(e_9) = FC(e_9)\,\Delta\,FC(e_1)$

(a) The tree transformation on the constraint graph.



(b) The calculation of coefficient matrix B' from B.



(c) The calculation of parity matrix P' from P.

Figure 3.9: The example of the pivoting using pivoting matrix T.

3.5.4 Pivoting

The general *pivoting* of Simplex Method refers to the process that improving the current solution (if is not optimal) by moving a nonbasic variable (entering variable) into the basis, and moving a basic variable (leaving variable) out of the basis. Similarly, the **pivoting** of the proposed matrix formulation is defined as:

Definition 10: A **pivoting** refers to the process that moving a non-tree edge variable e_{c_p} out of the basis, and moving a tree-edge variable e_{t_q} into the basis, where $b_{pq} = 1$ (b_{pq} is the entry of the coefficient matrix B). A pivoting between leaving variable e_c and entering variable e_t is denoted as $PV(e_c, e_t)$.

The definition specifies that the tree edge e_{t_q} must be in the fundamental loop of the non-tree edge e_{c_p}, or the non-tree edge e_{c_p} must be in the fundamental cutset of the tree edge e_{t_q}, which is equivalent to an elementary tree transformation.

To perform a pivoting, a *pivoting matrix* T is introduced as:

$$T \otimes B \otimes E = T \otimes P \tag{3.24}$$

T is a $n_c \times n_c$ square matrix written as:

$$T = \begin{array}{c} \begin{array}{ccccccc} t^1 & \cdots & & t^p & & \cdots & t^{n_c} \end{array} \\ \begin{pmatrix} 1 & & & b_{1q} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & \vdots & & & \\ & & & b_{pq} & & & \\ & & & \vdots & 1 & & \\ & & & \vdots & & \ddots & \\ & & & b_{n_c q} & & & 1 \end{pmatrix} \end{array} \tag{3.25}$$

w , i.e., $t^p = b^q$.

The matrix B' after the pivoting is calculated as:

$$B' = T \otimes B = (b'^1 \ \cdots \ b'^p \ \cdots \ b'^q \ \cdots \ b'^n)^T \tag{3.26}$$

w

$$' = \begin{cases} , & = \ ; \\ = \quad , & = \ . \end{cases} \tag{.27}$$

And the matrix P' after the pivoting is calculated as:

$$P' = T \otimes P = \begin{cases} P, & p_p = 0; \\ P \oplus b^q, & p_p = 1. \end{cases} \tag{3.28}$$

This calculation can be easily proved given the definition of $\oplus$ and $\otimes$ operations, so the detailed steps are omitted.

Fig.3.8 shows an example of a pivoting $PV(e_3, e_5)$ using a pivoting matrix T. In this example the entering variable is e_5 and the leaving variable is e_3.

According the definition of v and e in Eq.3.14 and Eq.3.15, the definition of the fundamental cutset vector exactly agrees with the the column vector b^i, and the definition of edge status vector Ψ also agrees with the matrix P. Therefore from Eq.3.27 and Eq.3.28, there are:

$$FC'(e_i) = \begin{cases} FC(e_i), & e_i \in FL(e_{c_p}); \\ FC(e_i) \triangle FC(e_{t_q}), & e_i \notin FL(e_{c_p}). \end{cases} \tag{3.29}$$

$$\Psi' = \begin{cases} \Psi, & p(e_{c_p}) = 0; \\ \Psi \triangle FC(e_{t_q}), & p(e_{c_p}) = 1. \end{cases} \tag{3.30}$$

The above Eq.3.29 and Eq.3.30 are exactly the same as the elementary tree transformation properties discussed in Section 3.4.2, and the examples are shown in Fig.3.9. Fig.3.9(a) shows the example of the elementary transformation between edges e_1 and e_7, and Fig.3.9(b) and Fig.3.9(c) show the corresponding matrix calculations.

Fig.3.9(b) i

-tree edge e_{c_p}, its fundamental cutset keeps unchanged, say edge e_4; otherwise, the fundamental cutset is updated using $\triangle$ operations between two vectors, as the columns for edges e_2, e_5 and e_9.

Fig.3.9(c) i

( ) =

( ) =

(,) ( ) = , it is called as an *effective pivoting*, otherwise it is called an *ineffective pivoting*. Ignoring all the ineffective pivots, all the pivotings mentioned are effective ones.

Theorem 3: For pivotings $PV_1(e_{c_1}, e_t)$ and $PV_2(e_{c_2}, e_t)$ with the same entering variable e_t, $\Psi'_1 = \Psi'_2$; the two pivotings are regarded as *equivalent* in terms of the edge status vector Ψ'.

From Theorem 3 it is known that for a specified entering variable e_t, all the pivotings $PV(e_c, e_t)$ where $e_c \in FC(e_t)$ are equivalent and undistinguished; so pivoting is denoted as as $PV(x, e_t)$ if the leaving variables for an entering variable e_t are not cared.

3.5.5 Pivot Selection

Pivot selection is a crucial step in the Simplex Method since good choices can lead to a significant speed up in finding the optimal solutions. However in the proposed formulation, a pivoting is not guaranteed to always improve the solution, so that in order to make a pivot selection that results in

Algorithm 4 General Pivot Selection

Ensure: Initial matrix formulation $B_o \otimes E_o = P_o$

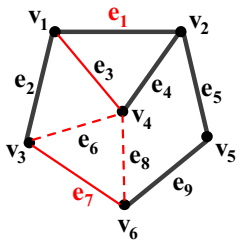
Require: Pivoting $PV(e_x, e_y)$ to be performed

1: Build initial edge status vector ψ and FC vectors

2: Calculate Edge Status Table for each tree edge e_t

3: Choose $PV(x, e_i)$ with the smallest $\|\psi'\|$ in Edge Status Table, $e_y = e_i$

4: Choose $e_j \in FC(e_i)$ randomly, $e_x = e_j$



Edge Status Table – Level 1

ψ_1	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\psi_1\|$
PV(x,e$_1$)	1					1				2
PV(x,e$_2$)		1	1			1				3
PV(x,e$_4$)				1		1	1	1		4
PV(x,e$_5$)			1		1			1		3
PV(x,e$_9$)			1					1	1	3

Figure 3.10: Pivot selection using an edge status table.

the smallest number of odd non-tree edges, all potential pivotings are needed to be checked, which is obviously time consuming.

Notice that each pivoting $T \otimes B \otimes E = T \otimes P$ consists two steps: the calculation of coefficient matrix $B' = T \otimes B$, and parity matrix $P' = T \otimes P$. Obviously the size of matrix B is much larger than P, and the calculation of B' is more time consuming than P'. Therefore it is expected that P' can be calculated first without the calculating B'.

Based on this motivation, an **edge status table** is proposed, where each row of the table is an edge status vector obtained after one pivoting $PV(x, e_{t_i})$. Fig.3.10 shows an example of the edge status table which shows the possible pivotings for all the tree edges. Then a pivoting selection method based on the edge status table is proposed shown in Algorithm 2. Before each pivoting, the edge status table is first calculated. For the entering variable, e_i of pivoting $PV(x, e_i)$ which has smallest $\|\psi\|$ is chosen; for the leaving variable, one variable $e_j \in FC(e_i)$ is randomly chosen.

3.5.6 Successive Pivotings

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\psi\|$
PV$_1$(e$_1$, e$_7$)+PV$_2$(e$_7$, e$_3$)	1					1				2
PV(e$_1$, e$_3$)	1					1				2
PV$_1$(e$_1$, e$_3$)+PV$_2$(e$_6$, e$_4$)				1		1	1	1		4
PV(x, e$_4$)				1		1	1	1		4

Figure 3.11: Examples of redundant successive pivotings.

Definition 12: Two adjacent pivotings, for example pivoting PV_1 is performed and is followed by

Edge Status Table – Level 2

PV_1	PV_2	ψ_2	x	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	
$PV_1(x,e_1)$	$PV_2(y,e_2)$	$\psi_1 \Delta FC(e_2)$	e_3	1	1					1			Need calculation
	$y \neq e_1$	$\psi \Delta FC(e_2)$	e_7		1	1			1				$= PV_1(x,e_2)$
	$PV_2(y,e_4)$	$\psi_1 \Delta FC(e_4)$	e_7			1				1			Need calculation
	$y \neq e_1$	$\psi \Delta FC(e_4)$	e_3				1		1	1	1		$= PV_1(x,e_4)$
	$PV_2(y,e_5)$	$\psi_1 \Delta FC(e_5)$	e_3	1					1				Need calculation
	$y \neq e_1$	$\psi \Delta FC(e_5)$	e_7			1		1			1		$= PV_1(x,e_5)$
	$PV_2(y,e_9)$	$\psi_1 \Delta FC(e_9)$	e_3	1					1	1	1	1	Need calculation
	$y \neq e_1$	$\psi \Delta FC(e_9)$	e_7			1					1	1	$= PV_1(x,e_9)$
$PV_1(x,e_2)$	$PV_2(y,e_4)$	$\psi_1 \Delta FC(e_4)$	e_7		1		1				1		Need calculation
	$y \neq e_2$	–	–										
	$PV_2(y,e_5)$	–	–										
	$y \neq e_2$	$\psi_1 \Delta FC(e_5)$	e_7				1		1			1	$= PV_1(x,e_5)$
	$PV_2(y,e_9)$	–	–										
	$y \neq e_2$	$\psi \Delta FC(e_9)$	e_7				1					1	1	$= PV_1(x,e_9)$

Figure 3.12: Example of the edge status table of level 2.

pivoting PV_2, are regarded as *successive pivotings*, denoted as $PV_1 + PV_2$. The original edge status vector is denoted as ψ, the one after PV_1 is ψ_1, and the one after PV_2 is ψ_2.

When applying Algorithm 2 for optimization, in each iteration, all possible pivotings are calculated; however there are some redundant calculation between two successive pivotings:

- The result of two successive pivotings may be the same as before pivoting, for example in Fig.3.11, the edge status vector ψ after $PV(e_1, e_3)$ is the same as the vector after two successive pivotings $PV_1(e_1, e_7) + PV_2(e_7, e_3)$;
- The result of two successive pivotings may have already been calculated: for example the ψ after $PV_1(e_1, e_3) + PV_2(e_6, e_4)$ is the same as $PV(x, e_4)$.

Therefore, for two successive pivotings, the following two theorems are proposed, to avoid redundant pivoting calculations in early stage, before the first pivoting being performed.

Theorem 4: For two successive pivotings $PV_1(e_x, e_y)$ followed by $PV_2(e_y, e_z)$, they have:

$$PV_1(e_x, e_y) + PV_2(e_y, e_z) = PV(e_x, e_z) \tag{3.31}$$

This theorem can be easily explained by tree transformation so that the proof is omitted.

Theorem 5: For successive pivotings $PV_1(e_{p_1}, e_{q_1})$ followed by $PV_2(e_{p_2}, e_{q_2})$, where $p(e_{p_1}) = 1$ and $p(e_{p2}) \oplus b_{p_2 q_1} = 1$: they have:

- w = , =
  = ( ) ( ))
- when $b_{p_1 q_2} = 1$, $b_{p_2 q_2} \oplus b_{p_2 q_1} = 1$:

$$\psi_2 = \psi \triangle FC(e_{q_2}) \tag{3.33}$$

- others: successive pivotings cannot be applied

Proof. Firstly, the condition $p(e_{p_1}) = 1$ and $p(e_{p2}) \oplus b_{p_2q_1} = 1$ are to guarantee that both PV_1 and PV_2 are effective pivotings, which are the same as the conditions in Eq.3.30.

Secondly, the condition that pivoting PV_2 can be performed is that $b'_{p_2q_2} = 1$, which refers to the entry of $b_{p_2q_2}$ after PV_1. When $b'_{p_2q_2} = 1$ is ensured, there are:

$$\psi_1 = \psi \triangle FC(e_{q_1}) \tag{3.34}$$

$$\psi_2 = \psi_1 \triangle FC'(e_{q_2}) \tag{3.35}$$

$$FC'(e_{q_2}) = \begin{cases} FC(e_{q_2}) \triangle FC(e_{q_1}), & b_{p_1q_2} = 1; \\ FC(e_{q_2}), & b_{p_1q_2} = 0. \end{cases} \tag{3.36}$$

$$\therefore \psi_2 = \begin{cases} \psi \triangle FC(e_{q_2}) \triangle FC(e_{q_1}), & b_{p_1q_2} = 1; \\ \psi \triangle FC(e_{q_2}), & b_{p_1q_2} = 0. \end{cases} \tag{3.37}$$

Since $b'_{p_2q_2}$ is calculated as:

$$b'_{p_2q_2} = \begin{cases} b_{p_2q_2}, & b_{p_1q_2} = 0; \\ b_{p_2q_2} \oplus b_{p_1q_2}, & b_{p_1q_2} = 1. \end{cases} \tag{3.38}$$

to ensure $b'_{p_2q_2} = 1$, there shall be $b_{p_1q_2} = 0,\ b_{p_2q_2} = 1$, which is the condition for Eq.3.32, or there shall be $b_{p_1q_2} = 1,\ b_{p_2q_2} \oplus b_{p_1q_2} = 1$, which is the condition for Eq.3.33. □

From Eq.3.32 and Eq.3.33 in Theorem 5, it shows that when performing two successive pivotings PV_1 and PV_2, the edge status vector ψ_2 of the PV_2 can be directly calculated using the original fundamental cutset vectors like $FC(e_{q_1})$ and $FC(e_{q_2})$. It means the result of two successive pivotings $PV_1 + PV_2$ can be calculated without actually performing pivoting PV_1. Besides, Eq.3.33 shows that, in some cases the successive pivotings $PV_1 + PV_2$ may get the same result as PV_1, and the calculations for these cases can be avoided.

Based on Theorem 4 and 5, an **edge status table of level 2** is proposed, to calculate the results of the successive pivotings based on the edge status table of level 1. Fig.3.12 shows an example of edge status table of level 2. Given the edge status table of level 1, the successive pivotings of each two tree edges e_{q_1} and e_{q_2} (order is neglected) are calculated, denoted as $PV_1(e_{p_1}, e_{q_1})$ and $PV_2(e_{p_2}, e_{q_2})$. For each PV_1, according to Eq.3.32 and Eq.3.33, there are two possible solutions: If $b_{p_1q_2} = 1$, like shown in the first row where e_{p_1} is e_3, the ψ_2 is calculated as $\psi_2 = \psi \triangle FC(e_1) \triangle FC(e_2)$ according to Eq.3.32; while if $b_{p_1q_2} = 0$, like shown in the second row where e_{p_1} is e_7, the ψ_2 is calculated as $\psi_2 = \psi \triangle FC(e_2)$ according to Eq.3.33. Moreover, in the second situation, the result of ψ_2 has already been calculated in the edge status table of level 1, so that no calculation is needed. It can be seen from the table shown

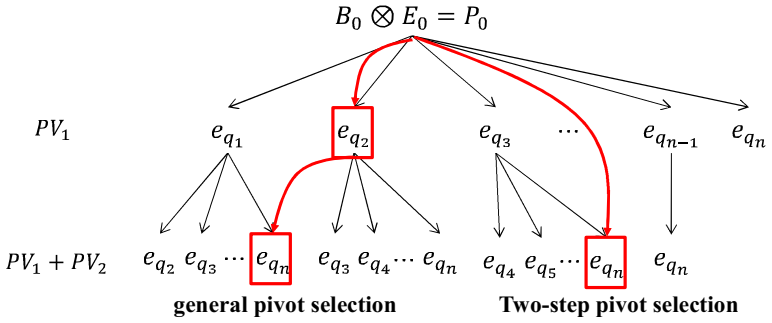


Figure 3.13: The two-step pivot selection

in Fig.3.12 that for all possible successive pivotings only a portion of them need calculation, and most successive pivotings can be skipped. Therefore the pivot selection efficiency is expected to be greatly enhanced.

3.5.7 A Two-Step Pivot Selection

Based on the edge status tables, a two-step pivoting selection method is proposed, shown in Algorithm 3.

Algorithm 5 Two-Step Pivot Selection

Ensure: Initial matrix formulation $B_0 \otimes E_0 = P_0$

Require: Pivotings PV_1 and PV_2 to be performed

1: Build initial edge status vector ψ

2: **Step 1:** Calculate Edge Status Table of Level 1

3: **Step 2:** Calculate Edge Status Table of Level 2

4: Choose k successive pivotings with the smallest $\|\psi_2\|$ in Edge Status Table of Level 2, solve the exact minimum vertex cover

5: Perform pivoting PV_1 and pivoting PV_2 with the smallest vertex cover solution

The major improvements from Algorithm 2 including:

- Determining two successive pivotings in one iteration is more efficient than that in Algorithm 2, because the execution of a pivoting is time consuming due to the large size of coefficient matrix B;
- It improves the optimality of the pivoting selections, because the selection of PV_1 may be blind if the result of PV_2 are unpredictable;

Fig.3.13 shows the major differences of the two pivot selection methods. Starting from a matrix formula $B_0 \otimes E_0 = P_0$, the two-step pivoting selection is able to forecast the results of $PV_1 + PV_2$ and skip the calculation of PV_1, and choose a best solution of $PV_1 + PV_2$.

Table 3.1: Information of Test Benches

DFG	V	E	$T_{lp}{}^{*}$	**DFG**	V	E	$T_{tp}{}^{*}$
arf	29	42	11	rand0	92	136	17
ewf	35	55	17	rand1	158	234	19
ar	29	42	11	rand2	632	943	30
ae	54	143	60	rand3	658	1392	42
ad2	47	110	47	rand4	917	2601	189
ellip	35	67	25	rand5	1916	5387	251
mpeg	54	114	36	rand6	1832	5428	583
fft	134	234	20				

* The longest path in DFG

3.5.8 Multi-Step Pivotings

In Section 3.5.6 two-step successive pivoting is discussed; in this section multi-step successive pivotings are discussed.

For successive pivotings $PV_1(e_{p_1}, e_{q_1})$, $PV_2(e_{p_2}, e_{q_2})$ and $PV_3(e_{p_3}, e_{q_3})$ with edge status vectors ψ (before PV_1), ψ_1, ψ_2 and ψ_3. Similar to Eq.3.33 and Eq.3.32, ψ_3 can be computed directly from ψ and $FC(e_{q_1})$ to $FC(e_{q_3})$ under some special conditions as:

$$\psi_3 = \begin{cases} \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_1}) \triangle FC(e_{q_2}), & \textit{cond.}\ 1; \\ \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_2}), & \textit{cond.}\ 2; \\ \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_1}), & \textit{cond.}\ 3; \\ \psi \triangle FC(e_{q_3}), & \textit{cond.}\ 4; \\ \textit{succesive pivotings cannot be performed}, & \textit{others}; \end{cases} \tag{3.39}$$

where *cond.* 1 is:

$$b_{p_1q_2} = 0,\ b_{p_1q_3} = 0,\ b_{p_2q_3} = 0,\ b_{p_2q_2} = 1,\ b_{p_3q_3} = 1 \tag{3.40}$$

and conditions *cond.* 2 to *cond.* 4 are more complicated. Besides, the conditions to guarantee PV_1 to PV_3 are all effective pivotings, similar to the ones in Theorem 5, are also needed. If all conditions are not met, successive pivotings cannot be applied. The proof is omitted due to page limit.

Knowing from Eq.3.39 and Eq.3.40, first, the conditions for three-step successive pivoting is much more strict than two-step pivoting as Eq.3.40, so that the situations that three-step pivotings cannot be applied are much more than two-step pivotings. Second, the condition check, i.e., the calculation of *cond.* 1 to *cond.* 4 is also much more time consuming than two-step pivotings. Consequently, it is believed that multi-step pivotings do not contribute to optimality more than two-step pivoting, and the time consumption will be much larger.

3.6 Experimental Results

The proposed algorithms are implemented in C on the platform of Windows7 with 2.8GHz CPU and 8GB memory. The algorithms are evaluated on randomly generated constraint graphs and real high

Table 3.2: Area, power and delay of multiplexers from [63].

MUX	Power(μW)	Area(μm^2)	Delay(ns)
2 to 1	27.58	130	0.81
4 to 1	44.24	317	1.05
8 to 1	78.82	748	1.40
16 to 1	138.99	1315	1.74
32 to 1	257.82	2464	2.08

T .3: comparison between elementary tree transformation based and matrix transformation based optimization methods

TestBench		**ILP**		**Initial Solution**			**T**				**Matrix Based Opt. I**[1]				**Matrix Based Opt. II**[2]			
Vtx.	Den.	INT	Cmp	INT	Cmp	T	INT	Cmp	Time	Cmp	INT	Cmp	Time	Cmp	INT	Cmp	T	Cmp
40	2.0	9	1	10.77	1.197	0.027	9.01	1.001	9.281	4.57	9.054	1.006	2.03	1	9.01	1.001	2.55	1.26
40	5.0	18	1	20.69	1.149	0.011	18.19	1.011	46.81	6.55	18.03	1.002	7.15	1	18.02	1.001	7.98	1.12
50	2.5	12	1	15.10	1.258	0.050	12.16	1.013	24.32	2.48	12.14	1.012	9.82	1	12.04	1.003	8.65	0.88
50	3.0	16	1	18.42	1.151	0.063	16.04	1.003	32.06	3.87	16.04	1.002	8.28	1	16.02	1.001	8.72	1.05
50	4.0	18	1	21.94	1.219	0.268	18.62	1.034	64.93	4.91	18.47	1.026	13.23	1	18.39	1.022	20.71	1.57
70	2.0	14	1	17.78	1.270	0.192	14.04	1.003	24.59	6.60	14.02	1.002	3.88	1	14.01	1.001	4.28	1.10
70	3.0	20	1	26.30	1.315	0.489	20.96	1.048	109.03	3.64	20.81	1.041	29.99	1	20.54	1.027	35.55	1.19
70	4.0	26	1	31.49	1.211	0.688	26.88	1.034	154.73	4.51	26.17	1.007	34.31	1	26.04	1.002	38.20	1.11
100	2.0	17	1	23.91	1.406	0.488	18.15	1.068	105.03	6.89	17.86	1.051	15.24	1	17.58	1.034	17.44	1.14
100	3.0	29	1	37.53	1.294	0.950	32.09	1.107	213.65	9.06	30.87	1.064	23.58	1	30.56	1.054	28.59	1.21
100	4.0	36	1	44.42	1.234	1.387	39.43	1.095	316.97	9.99	38.11	1.059	31.72	1	37.77	1.049	39.02	1.23
200	2.0	32*	1	47.37	1.480	1.94	40.12	1.254	163.85	2.55	35.62	1.113	64.25	1	35.28	1.103	63.16	0.98
200	4.0	77*	1	87.66	1.138	4.75	84.04	1.091	529.66	5.13	80.86	1.050	103.29	1	77.92	1.082	92.28	0.89
300	2.5	72*	1	91.60	1.272	5.14	83.45	1.159	636.07	6.25	80.90	1.124	101.80	1	78.01	1.083	122.99	1.21
300	4.5	129*	1	140.67	1.090	11.80	135.15	1.048	1439.5	6.15	135.34	1.049	233.98	1	133.27	1.033	298.00	1.27
400	2.5	104*	1	124.26	1.195	7.10	114.35	1.100	1170.9	5.88	111.03	1.068	198.99	1	109.05	1.049	257.98	1.30
400	4.5	174*	1	190.91	1.097	15.49	185.60	1.067	2658.5	6.03	181.51	1.043	440.58	1	177.45	1.020	576.44	1.31
500	5.0	219*	1	253.22	1.156	24.29	249.10	1.137	5462.6	6.57	243.18	1.110	831.23	1	226.05	1.032	980.51	1.18
600	5.0	285*	1	304.55	1.069	43.57	300.20	1.053	9866.0	8.00	292.99	1.028	1233.3	1	289.05	1.014	1260.2	1.02
AVG			**1**		**1.221**			**1.069**		**5.77**		**1.045**		**1**		**1.032**		**1.16**

[1] The general pivot selection shown in Algorithm 2 is used in the local search optimization.

[2] The two-step successive pivot selection shown in Algorithm 3 is used in the local search optimization.

* For testcases with more than 200 vertices the ILP failed to produce solutions within limited time. The values here are the minimum numbers of interconnections among 1000 runs produced by the proposed algorithm.

Table 3.4: port assignment on high level synthesis testbenches

Test	W/O PA		Greedy[54]		T		Matrix Based Opt. I		Matrix Based Opt. II	
Bench	INT	Time	# of INT	Time(ms)	# of INT	Time(ms)	# of INT	Time(ms)	# of INT	Time(ms)
arf	19	4	18(94.74%)	+0.022(0.55%)	18(94.74%)	+0.39(9.75%)	17(89.47%)	+0.17(4.25%)	17(89.47%)	+0.18(4.50%)
ewf	17	4	16(94.12%)	+0.021(0.53%)	15(88.24%)	+0.67(16.75%)	16(94.12%)	+0.14(3.50%)	15(88.24%)	+0.15(3.75%)
ar	31	6	28(90.32%)	+0.019(0.32%)	28(90.32%)	+0.41(6.83%)	28(90.32%)	+0.07(1.17%)	28(90.32%)	+0.07(1.17%)
ae	35	16	32(91.43%)	+0.048(0.30%)	31(88.57%)	+0.87(5.44%)	31(88.57%)	+0.31(1.94%)	27(77.14%)	+0.22(1.38%)
ad2	37	16	33(89.19%)	+0.035(0.22%)	31(83.78%)	+1.41(8.81%)	31(83.78%)	+0.22(1.38%)	30(81.08%)	+0.29(1.81%)
ellip	45	14	42(93.33%)	+0.019(0.14%)	40(88.89%)	+0.82(5.86%)	40(88.89%)	+0.05(0.36%)	40(88.89%)	+0.05(0.36%)
mpeg	35	20	32(91.43%)	+0.055(0.28%)	30(85.71%)	+1.00(5.00%)	29(82.86%)	+0.10(0.50%)	29(82.86%)	+0.11(0.55%)
fft	75	46	67(89.33%)	+0.095(0.21%)	63(84.00%)	+10.01(21.76%)	63 (84.00%)	+4.44(9.65%)	62(82.67%)	+4.52(9.83%)
rand0	81	21	74(91.36%)	+0.048(0.23%)	73(90.12%)	+1.15(5.48%)	74(91.36%)	+0.40(1.90%)	72(88.89%)	+0.39(1.86%)
rand1	129	54	123(95.35%)	+0.053(0.10%)	120(93.02%)	+4.47(8.28%)	118(91.47%)	+1.09(2.02%)	118(91.47%)	+1.28(2.37%)
rand2	458	371	418(91.27%)	+0.591(0.16%)	409(89.30%)	+92.70(24.99%)	405(88.43%)	+17.24(4.65%)	402(87.77%)	+19.69(5.31%)
rand3	554	260	513(92.60%)	+1.81(0.70%)	487(87.91%)	+82.18(31.61%)	485(87.55%)	+9.09(3.50%)	480(86.64%)	+12.86(4.95%)
rand4	692	417	654(94.51%)	+2.05(0.49%)	630(91.04%)	+48.77(11.70%)	623(90.03%)	+15.02(3.60%)	621(89.74%)	+18.09(4.34%)
rand5	1158	2137	1030(88.95%)	+5.84(0.27%)	1020(88.08%)	+237.8(11.13%)	1019(88.00%)	+88.50(4.14%)	1006(78.50%)	+94.22(4.41%)
rand6	1113	2088	1068(95.96%)	+12.14(0.58%)	1044(93.80%)	+426.2(20.41%)	1033(92.81%)	+108.2(5.18%)	1031(92.63%)	+121.7(5.83%)
AVG			**92.26%**	**+0.34%**	**89.17%**	**+12.92%**	**88.78%**	**3.18%**	**86.42%**	**+3.49%**

T .5: Estimated area, power and critical path delay of all MUXes.

Test	W/O PA			Greedy[54]			MINE			
Bench	Power	Area	Delay	Power	A	Delay	Power	Area	Delay(w/o cp)	Delay(with cp)
arf	256.29	2030	1.05	247.78 (96.68%)	1937 (95.42%)	1.05 (100.0%)	239.31 (93.37%)	1829 (90.10%)	1.05 (100.0%)	0.93 (88.57%)
ewf	220.37	1807	1.05	211.73 (96.08%)	1699 (94.23%)	0.93 (88.57%)	203.26 (92.24%)	1591 (88.05%)	0.93 (88.57%)	0.93 (88.57%)
ar	356.16	3198	1.14	322.48 (90.54%)	2949 (92.21%)	1.05 (92.11%)	322.48 (90.54%)	2949 (92.21%)	1.05 (92.11%)	1.05 (92.11%)
ae	391.85	3666	1.23	367.05 (93.67%)	3380 (92.20%)	1.23 (100.0%)	324.00 (82.68%)	2841 (77.50%)	1.14 (92.68%)	1.05 (85.37%)
ad2	406.89	3808	1.44	374.57 (92.06%)	3451 (90.63%)	1.23 (85.42%)	349.76 (85.96%)	3164 (83.09%)	1.23 (85.42%)	1.14 (79.17%)
ellip	416.60	3949	1.53	394.12 (94.60%)	3735 (94.58%)	1.49 (97.39%)	379.17 (91.02%)	3593 (90.99%)	1.44 (94.12%)	1.31 (85.62%)
mpeg	411.47	3697	1.23	385.68 (93.73%)	3388 (91.64%)	1.23 (100.0%)	359.92 (87.47%)	3165 (85.61%)	1.23 (100.0%)	1.05 (85.37%)
fft	720.84	6830	1.49	660.77 (91.67%)	6262 (91.68%)	1.49 (100.0%)	622.14 (86.31%)	5870 (85.94%)	1.44 (96.64%)	1.44 (96.64%)
rand0	867.57	8149	1.31	812.68 (93.67%)	7579 (93.01%)	1.31 (100.0%)	797.63 (91.94%)	7437 (91.26%)	1.23 (93.89%)	1.05 (80.15%)
rand1	1151.00	10930	1.66	1106.34 (96.12%)	10510 (96.16%)	1.61 (96.99%)	1069.01 (92.88%)	10153 (92.89%)	1.61 (96.99%)	1.57 (94.58%)
rand2	4489.64	42507	1.78	4183.37 (93.18%)	39486 (92.89%)	1.76 (98.87%)	4062.10 (90.47%)	38313 (90.13%)	1.74 (97.75%)	1.61 (90.45%)
rand3	5416.90	51224	1.87	5106.87 (94.28%)	48238 (94.17%)	1.83 (97.86%)	4854.54 (89.62%)	45748 (89.31%)	1.80 (96.26%)	1.76 (94.12%)
rand4	6454.39	61163	2.02	6170.18 (95.60%)	58453 (95.57%)	1.95 (96.53%)	5923.31 (91.77%)	56092 (91.71%)	1.91 (94.55%)	1.85 (91.58%)
rand5	10254.15	97466	2.08	9301.22 (90.71%)	88288 (90.58%)	2.02 (97.12%)	9122.93 (88.97%)	86569 (88.82%)	2.00 (96.15%)	1.89 (90.87%)
rand6	9919.28	94226	2.04	9584.13 (96.62%)	91014 (96.59%)	2.02 (99.02%)	9308.06 (93.84%)	88346 (93.76%)	1.95 (95.59%)	1.87 (91.67%)
AVG	**1**	**1**	**1**	**93.94%**	**93.41%**	**96.65%**	**89.93%**	**88.75%**	**94.71%**	**88.98%**

level synthesis benchmarks from[27].

3.6.1 Optimality and Runtime Test on Random Constraint Graph

In Table 4.3, the proposed algorithms are first evaluated on the randomly generated constraint graphs with different numbers of vertices and graph densities. The column Vtx. shows the number of vertices of the constraint graph from 40 to 600, and the column Den. shows the graph density varying from 2.0 to 5.0. For each pair of Vtx. and Den., one graph is randomly generated as the constraint graph. On each constraint graph, the proposed algorithm is executed for 1000 times and the number of interconnections is the average value of 1000 runs. The # of INT shows the total number of interconnections. The columns of Cmp refer to the comparisons.

The proposed algorithms are compared to the optimum solutions generated by Integer Linear Programming, shown in the column of ILP, solved by *lp_solver*[56], including: 1) the initial solutions, obtained by conflict graph and minimum vertex cover discussed in Section 3.3, shown in the column of Initial Solution; 2) the elementary tree transformation based Algorithm 1, discussed in Section 3.4, shown in the column of Tree Based Opt.; 3) the matrix based optimization discussed in Section 3.5, where the general pivoting selection of Algorithm 2 is adopted, shown in the column of Marix Based Opt.I; 4) the matrix based optimization, where the two-step pivoting selection of Algorithm 3 is adopted, shown in the column of Matrix Based Opt.II.

In terms of solution optimality, i.e., the number of interconnects, comparing to ILP, it shows that the initial solutions are 1.221, the tree based optimization solutions are 1.069, the matrix based optimization solutions using general pivot selection are 1.045, and the solutions using two-step pivot selection are 1.032 times overhead, respectively. The matrix based optimization using two-step pivot selection shows highest optimality, because it is able to predict two successive pivotings in one iteration.

In terms of algorithm efficiency, the execution times of three optimization algorithms are shown in milliseconds. The matrix based optimization using general pivoting is taken as the baseline for comparison since it shows the shortest execution time. The elementary transformation based optimization runs 5.7 times slower, because the tree transformations consume most of the execution time. The two-step pivoting optimization has a slight overhead compared to the general pivoting optimization, 1.16 on average. It is mainly because of the condition checks in Eq.3.33 and Eq.3.32. Even though, compared to the elementary tree transformation based optimization, the matrix based optimization still shows 4.9 times speedup.

3.6.2 Optimality and Runtime Test on DFGs

The proposed port assignment algorithms are also applied on real high level synthesis benchmarks, by comparing to the previous work[54] and[36], shown in Table 3.4. The benchmarks are given in the format of data flow graphs (DFG) as shown in Table I. The first eight benchmarks are taken from[27], and benchmarks from rand0 to rand6 are randomly generated using a DFG generator *TGFF*[62]. Each benchmark is given the number of vertices in Column *V*, the number of edges in Column *E*, and

the length of longest path in DFG is shown in Column T_{lp}. The scheduling, functional unit (FU) and register binding are first performed using the method in[27]; the number of interconnections is shown in Column *INT* with runtime measured in milliseconds. Then port assignment is performed to minimize the number of interconnections. The Greedy[54] shows the results of the algorithm proposed in[54]; The FM[3] shows the results of the FM based algorithm proposed in[36]. The Tree Based Opt. shows the results of the algorithm proposed in[33] and[34], in which the elementary tree transformation is adopted for optimization. The Matrix Based Opt.I and Matrix Based Opt.II show the results of the proposed matrix based algorithm using pivotings, where Opt.I uses the general pivot selection shown in Algorithm 2, and Opt.II uses the two-step pivot selection shown in Algorithm 3.

It shows that, the greedy algorithm in[54] reduces the number of interconnections by 7.7%, with a time overhead of 0.34%; it shows high efficiency because of the algorithm simplicity. The FM based algorithm[36] reduces the number of interconnections by 9% with 2.06% time overhead. The elementary tree transformation based algorithm reduces the number of interconnections by 10%, which is slightly higher than FM but with a large time overhead of 13%. The matrix based optimization using general pivot selection, reduces the interconnections by 11.2% with 3.18% time overhead; the two-step pivot selection shows the highest optimality, reducing the interconnections by 13.7% with 3.49% time overhead.

The results show that the matrix method proposed in this work excels the FM based algorithm in terms of optimality, and the possible reasons could be:

1. The FM based algorithm in [3] is a local search based optimization method, where a bipartite graph is used for vertex partition. In each iteration of local search, on the bipartite graph, either one vertex is moved from one partition to the other, or two vertices are swapped to change their partitions. Thus, each time the number of vertices whose partitions are changed is limited to 1 or 2. It means when the solutions in the neighborhood are being searched by local search, only the ones that are quite close to the current solution are investigated; that is, the search area is very narrow. In the actual experiments, it show that, the possibility that a better solution exists nearby the current solution is low; as a result, the local search mostly terminates only after a few iterations, w

   ( ). This corresponds to the operation that changes the partitions of the descendant vertices of tree edge e_t on the graph. Thus, the number of vertices whose partitions are changed is much larger than that in [3]. It means that, in the matrix based algorithm, the solutions far from the current solution are also investigated, and the search area of my proposal is broader than [3]. Moreover, by proposing the successive pivoting method, the search area is further broadened by predicting the solutions of the next iteration. Benefiting from the diversification of search area, the matrix based algorithm shows higher optimality than [3].

Table 3.6: Implementation of testbench FFT on FPGA platform under 50MHz clock frequency

	Power(*mW*)			R	)	Delay(*ns*)	
	Total	*dyn*	*leak*	# of Reg.	# of LUTs	Max. Frq.[1]	Min. Prd.[2]
W/O PA	327	286	41	2282	3598	57.607	17.359
Greedy[54]	313	272	41	2092	3518	58.517	17.089
Mine w/o cp	289	249	40	1900	3453	60.140	16.628
Mine	289	249	40	1908	3446	60.190	16.614

[1] Max. Frq: maximum frequency (MHz)
[2] Min. Prd: minimum period (ns)

3.6.3 Power, Area and Delay Evaluation

In this section the estimated improvements of area, power and critical path delay achieved by the proposed port assignment algorithm are shown. Reducing the number of interconnections by port assignment leads to the reduction of MUX size before the input ports of functional units; smaller MUXes result in smaller area, lower power and shorter critical path delay. In this work, the reduction of area, power and critical path delay are estimated by calculating all the MUXes before input ports of FUs, because the proposed algorithm does not change the FU and register binding but only affects the MUX size. The area, power and delay information of MUXes of different sizes are obtained from work[63] and shown in Table II. Since only 2-to-1, 4-to-1, 8-to-1, 16-to-1 and 32-to-1 MUXes are given, they are combined to make MUXes of other sizes like 23-to-1. The total area and power are computed by summarizing all the MUXes in a benchmark, and the critical path delay is approximately estimated using the largest MUX in a benchmark.

Table 3.5 shows the estimated reduction of area, power and critical path delay of all MUXes after port assignment, by comparing to the results without port assignment. It shows that the work in[54] is able to reduce power by 6% and area by 6.6% of MUXes on average, and this work is able to reduce power by 10% and area by 11.2% on average. As for the critical path delay, the work in[54] reduces the delay by 3.5%, and the proposed one reduces the delay by 5.3% if no special optimization for critical path is conducted, as shown in the Column Delay(w/o cp). If the critical path is considered, the delay is reduced by 11% as shown in the Column Delay(with cp), achieved by reducing the size of the largest MUX.

Moreover, a most typical testbench FFT is picked up and implement it on the FPGA platform to evaluate the actual improvement in terms of power, area and delay. The simulation is conducted using Xillinx ISE 14.7 on the evaluation platform Spartan-6 SP605 Evaluation Platform under a clock frequency of 50MHz. The results are shown in Table VI, including the result without port assignment, the result produced by[54], and the result produced by the proposed algorithm without and with critical path consideration. It shows that the port assignment indeed reduces the power and area (reflected by the resources on FPGA), and increases the maximum clock frequency as well. Compared to the work in[54], the proposed algorithm achieves more power and area reduction and higher clock frequency.

3.7 Summary

In this work a port assignment algorithm for interconnect reduction after the FU and register binding in HLS is proposed. First the problem is formulated on a constraint graph, and feasible solutions are obtained by spanning tree and vertex cover. Then the solutions are optimized by an elementary tree transformation based iterative method. To speedup, the problem is again formulated by a matrix, and by substituting the $\oplus$ for $+$ operations in LP formulation, network simplex method are adopted with pivoting operations to perform optimization. The two-step successive pivoting is also discussed to further improve algorithm efficiency and optimality. The experimental results show that on the randomly generated test cases, the matrix-based port assignment algorithm shows higher optimality and efficiency than the spanning tree based algorithm, which is 4.9 times faster when both algorithms produce optimum solutions with a probability higher than 99%. On the real benchmarks, 14% interconnections are reduced with 3.5% time overhead on average, while the previous work reduces 8% on average.

4

A Multi-Level Algorithm for 3D-IC TSV Assignment in Physical Synthesis

In this chapter, the third topic, TSV insertion for 3D-IC, is to be discussed, which corresponds to the physical synthesis step. It determines the positions of TSVs on each TSV die, to minimize the total wire length. Comparing to the problems in HLS, it is a huge sized problem, so the purpose of this research is to propose a high efficiency algorithm, to solve the problem not only fast but without sacrificing the solution quality. This problem is first formulated as the Integer Multi-Commodity Min-Cost flow problem, and then an iterative multi-level algorithm is to be proposed to handle the huge sized inputs. To improve the solution quality, a mixed single and multi-commodity flow method is to be proposed, followed by a bipartite matching method for solution optimization. The detailed formulations and algorithms are discussed in the following of this chapter.

4.1 Introduction

Three-dimensional integrated circuit (3-D IC) is a promising IC manufactural technology with stacks of dies that approaches higher density, reduced power, smaller footprint, improved performance and lower cost compared with traditional monolithic ICs, and many important works on 3-D IC technology are proposed in recent years[65][66]. The 3-D IC technology uses through-silicon vias (TSV) to provide vertical electrical connections passing through a silicon wafer or die. The TSV is an emerging interconnection technology that will replace the traditional wire-bonding process in chip/wafer stacking, to increase inter-die communication bandwidth, reduce form factor, and lower power consumption of stacked multi-die systems by eliminating the need of long cross-chip interconnects existing in 2-D ICs.

One of the key design challenges of 3-D IC is the optimization of the number and locations of TSVs, which is generally called the TSV assignment problem. Given a 3-D IC netlist describing the inter-die nets, TSV assignment is to decide which TSVs are used to implement the nets spanning

different chip dies. After the TSV assignment, routing is applied on each die to complete the electrical connection of every net. Since the TSVs are very large compared to logic gates, say dozens to hundreds of times the area of a standard cell, the TSV assignment is crucial to the wire length and signal delays of 3-D circuits, and it is now attracting broad interest among both academic and industrial researchers.

There are already several literatures that have been proposed for the TSV assignment problem on 3-D IC[67] -[77]. In[68] the TSV assignment is considered during the routing process and a min-cost max-flow method is proposed. In[69] the inter-layer TSV placement is discussed, but the formulation is ideal, where TSVs are considered placable anywhere on the chip even the spaces occupied by modular. In[70] a two-stage 3-D floorplanning algorithm is proposed, where the first stage plans hard macros and TSV-blocks, and the second stage reassigns signal TSVs. Some more recent works tend to solve the TSV assignment net by net, for example the work in[72] proposes a minimum spanning tree (MST) based method which applies Kruskal's algorithm net by net. The work[76] proposes an integrated method for pre-placed TSVs on chip dies, which has four steps including net by net shortest path search, bipartite matching, min-cost max-flow and postprocessing. The work in[77] formulates the TSV assignment on the grid structure that, after the floorplan of blocks, the available grids for TSVs are calculated according to the white space, and the lagrangian relaxation is used to minimize the total wire length.

Some of the previous works, for example[76] and[77], formulated the TSV assignment problem as an *integer min-cost multi-commodity problem (IMCMC)*. In this work, the same IMCMC formulation is adopted, and a multi-level heuristic algorithm is proposed aiming at both high efficiency and optimality. The contributions of this proposal are as the following, among which (1) and (2) are briefly discussed in[64]:

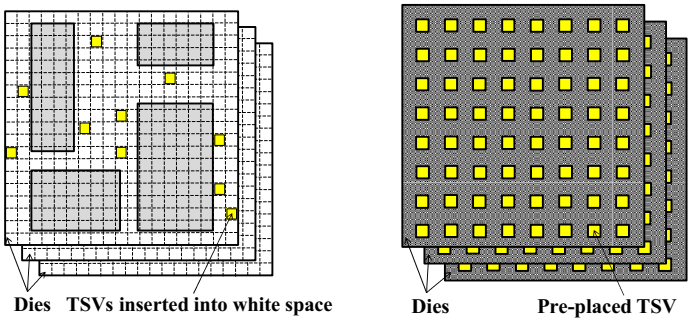
1. To reduce the IMCMC network complexity, i.e., the number of edges, **a multi-level method** is proposed including grid coarsening and un-coarsening. Benefiting from the multi-level proposal, the pure multi-commodity problem is transformed into **a mixed single and multi commodity problem**, and the net order reliance is partially removed.

2. Given an initial TSV assignment, an **extended layer by layer optimization** method is proposed to improve the solution quality; it extends the optimization from adjacent two chip layers to arbitrary two chip layers.

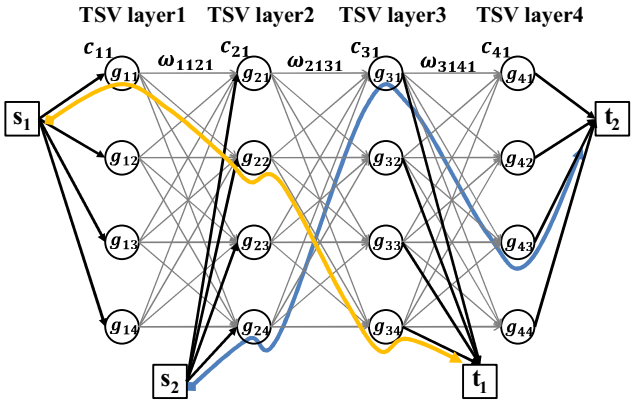
3. Since the number of generated edges in the coarsened graphs may be to limited to find a good TSV assignment, a **grid extension method** is proposed, extended from[64].

4. The mixed flow algorithm is also improved from[64] to enhance rough flow assignment optimality, including: (1) to decrease the routing detours, the formulation graph are modified to take **TSV congestions** into consideration; (2) the shortest path computation in the mixed flow algorithm is modified by introducing a **penalty function**; (3) the single flow algorithm is **applied on the grid vertices** of coarsened graphs.

5. In the experiments, comparing to previous work[76], the proposed algorithm reduces the wire length by 7.0% with a speedup of 37X. In additional, comparing to an earlier work[64], the wire length is reduced by 2.3%.



(a) W (b) Pre-placed TSVs



(c) Integer min-cost multi-commodity flow problem formulated on the IMCMC network

Figure 4.1: Both kinds of TSV assignment problems can be formulated as an integer min-cost multi-commodity (IMCMC) problem.

The remainder of the paper is organized as follows. Section 2 formulates the problem, Section 3 introduces the multi-level proposal, Section 4 introduces the mixed single and multi commodity flow algorithm, and Section 5 discusses the extended layer by layer optimization. Section 6 shows the experimental results and Section 7 concludes the paper.

4.2 Problem Formulation

4.2.1 Problem Description

There are basically two classes of TSV assignment. One is the grid structure like in[77 74 75], where chip dies are evenly divided into $P \times Q$ grids, and TSVs can only be inserted into white spaces among modules, as shown in Fig.4.1(a). The other one is the pre-placed TSVs[72 76], where the nets shall use the TSVs that are already inserted into the chip with fixed positions, as shown in Fig.4.1(b). The two models can be formulated in a unified way using the *capacity* to describe the number of TSVs that can be inserted, and using *grid* to represent both the grids in the grid structure and the pre-placed TSVs. The **inputs** include:

- A 3-D IC with n chip dies $D = \{d_1, d_2, \cdots, d_n\}$ and n-1 TSV layers $L = \{l_1, l_2, \cdots, l_{n-1}\}$, where TSV layer l_k connects chip dies d_k and d_{k+1}.
- The set of grids $g \in \Gamma$ representing the pre-placed TSVs or grids in the grid structure. Each grid g has a capacity c_g, referring to the number of TSVs that can be inserted into g. For the pre-placed TSVs, $c_g = 0$ or 1, and for the grids, $c_g \geq 0$. Each grid g is associated with a position from the left bottom corner of the chip, denoted as $p_g = (x_g, y_g, z_g)$, where z_g is the chip die of g.
- The 3-D net list with total m nets, denoted as NL, which describes the connections of pins. Each net is a two-pin net, denoted as $\eta = (s, t) \in NL$, where s is the source pin and t is the sink pin, gotten by multi-pin decomposing using the minimum spanning tree[76]. Each pin also has a position $p_s = (x_s, y_s, z_s)$ or $p_t = (x_t, y_t, z_t)$. z_s and z_t refer to the chip dies of s and t, where $z_s > z_t$.

The bounding box of a net $\eta = (s, t)$ is defined as a 2-D minimum rectangular region encapsulating both source pin s and sink pin t, which are projected onto one layer, because in this work the length of the vertical connection of a two-pin net is assumed as a constant. For a net $\eta = (s, t)$, $z_t - z_s$ TSVs are needed, denoted as τ_1 to $\tau_{z_t - z_s}$; the wire length of η, denoted as $wl(\eta)$, is estimated by accumulating the Manhattan Distances of TSVs in adjacent layers or source/sink pins:

$$wl(\eta) = d_{s,\ 1} + \sum_{1 \leq i < z_t - z_s} d_{\ i,\ i+1} + d_{\ z_t - z_s, t} + \cancel{(z_t - z_s)} \tag{4.1}$$

where $d_{p,q}$ is the Manhattan Distance of p and q: p and q refer to TSVs in adjacent layers or source/sink pins; $(z_t - z_s)$ represents the length of vertical connections, which is a constant and not considered in this work.

The **constraint** is, for each grid g, the number of TSVs in g occupied by nets is not larger than its capacity c_g.

The **goal** is to find a TSV assignment for each net $\eta_i = (s_i, t_i)$ to minimize the total wire length, computed as:

$$min: \sum_{1 \leq i \leq m} wl(\eta_i) \tag{4.2}$$

4.2.2 IMCMC Network Formulation

The TSV assignment problem is formulated as an Integer Multi-Commodity Min-Cost (IMCMC) flow problem on an **IMCMC network**[76], which is a directed graph $G = (V, E)$ with capacities and costs associated on edges. The vertices $V = V_g \cup V_s \cup V_t$, where $g_{mn} \in V_g$ represents the n-th grid on the m-th TSV layer, V_s represent the source pins and V_t represent the sink pins. Each grid vertex g_{mn} is associated with a capacity of c_g. The edges $E = \{e | e = (u, v)\}$, where $u \in V_s, v \in V_g$, or $u \in V_g, v \in V_t$, or $u, v \in V_g, d_v = d_u + 1$; edge direction is from u to v. Each edge (u, v) is associated with a cost ω_{uv}, defined as the routing cost between pins and TSVs, w

$d_{u,v}$, the same as in Eq.1. The capacities of edges coming from s or going into t are 1, and the capacities of the edges between grid vertices are infinite.

Assigning TSVs for a net $\eta = (s, t)$ is equivalent to sending a flow from s to t on the IMCMC network through grid vertices. The flow on edge $e = (u, v) \in E$ from s to t is denoted as f_{uv}^{st}. The objective is to find a flow for each net $\eta = (s, t)$ under the constraint of vertex capacities, to minimize the total cost of all the nets.

Fig.4.1(c) shows an example of IMCMC network with two nets $\eta_1 = (s_1, t_1)$ and $\eta_2 = (s_2, t_2)$. A flow from s_1 to t_1 going through three grids g_{11}, g_{22} and g_{34}; so that the three corresponding TSVs are assigned to Net1.

4.3 Previous Issues and Motivations

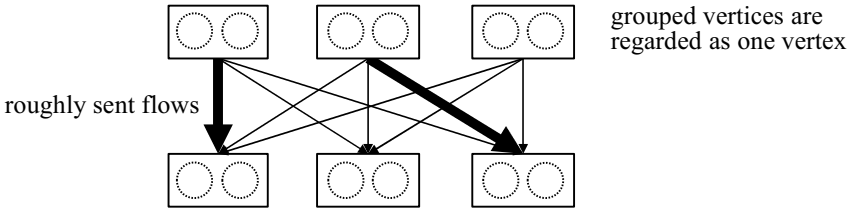
Given the IMCMC formulation, several problems are unavoidable and are already exposed in the existing works.

First, the complexity of a complete IMCMC network is extremely high. Assume a 3-D chip has n dies and m nets, and each die is divided into $P \times Q$ grids; the number of possible edges between grids is $(n-1) \cdot (P \times Q)^2$, and between source/sink pins and grids is $m \times P \times Q$. The total number of edges easily reaches millions, which greatly limits the efficiency of the proposed algorithms (eg. when $P = Q = 64$, $n = 4$, the number of edges exceeds 5 millions). In the work[77], the chip dies are divided into 80×80 for small testbench such as *ami49*, while for large testbenches such as *n100* and *n300*, the chip dies are only divided into 30×30 grids. This is because the algorithm is not able to handle more grids for larger testbenches, which degrades the accuracy of TSV assignment. The work in[76] reduces the number of possible paths by restricting that for a net $\eta = (s, t)$ going through TSV τ, the summation of the straight-line distance from s to τ and from τ to t does not exceed the length of the diagonal line of the chip die. However this method does not reduce the number of edges on the IMCMC network, and moreover the solution quality is sacrificed.

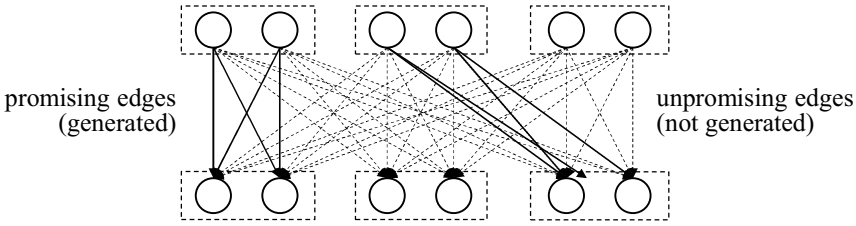
Second, because of the NP-completeness of the integer multi commodity problem, optimal solutions are extremely difficult to be obtained; once a flow is assigned for a commodity, it is fixed and cannot be updated. In this problem, most previous works assign the TSVs net by net; thus, the TSV assignment solution greatly relies on the net order, and may greatly deteriorate due to a bad order. To obtain stable and high quality solutions, the reliance of net order is highly expected to be removed.

Besides, the existing optimization methods are lack of efficiency. The post processing in[76] suffers from $100X$ to $800X$ runtime overhead for 0.2% solution improvement on average, and the Lagrangian Relaxation in[77] also suffers from $5X$ runtime overhead. Thus, a fast yet effective optimization algorithm is expected.

Motivated by these problems, the proposed algorithm first targets at reducing the number of edges of the IMCMC network. On the IMCMC network, only a small fraction of the edges are actually used; therefore it is not necessary to generate all but only the *promising edges* — the edges that have higher possibilities to be used in the final TSV assignment solution. Based on this observation, a **multi-**



(a) Flows are roughly sent on grouped vertices



( ) Only promising edges are generated according to the rough flows from (a) to reduce the total number of edges

Figure 4.2: Main idea of the proposed multi-level algorithm.

level algorithm is proposed, whose basic idea is shown in Fig.4.2. On the original IMCMC network, adjacent vertices together are grouped level by level to build smaller networks, and send rough flows between the coarsened vertices, called *rough flow assignment*, as shown in Fig.4.2(a). The edges on which rough flows are sent are regarded as *promising*, and are generated when the coarsened vertices are uncoarsened, as shown in Fig.4.2(b); the unpromising edges are not generated to reduce the total number of edges. The multi-level algorithm is to be introduced in Section 4.

4.4 Multi-Level TSV Assignment

The overall multi-level algorithm is shown in Algorithm 1, including three parts: the first part is **grid coarsening and source grouping** on the IMCMC network, to build **coarsened grids** of multiple levels, shown in line 1-10; the second and third parts are **rough flow assignment** and **graph uncoarsening**, which are executed by turns on each level of graphs composed of coarsened grids, as shown in line 11-15. Each time after rough flow assignment being performed, promising edges are generated on coarsened graphs according to the flow assignment result, shown in line 12.

4.4.1 Grid Coarsening and Source Grouping

First, the grid coarsening is performed level by level, and in each level, denoted as ε, the **coarsened grids** are generated:

Definition 1: A **coarsened grid of level** ε is defined as the grid clustered from $k_1 \times k_2$ grids of level $\varepsilon - 1$, denoted as grd_{ij} $(\varepsilon > 0)$. The capacity of $grid_{ij}$ is calculated as the capacity summation of all the $k_1 \times k_2$ grids of level $\varepsilon - 1$. The grids of original IMCMC network are level 0 $(\varepsilon = 0)$.

Algorithm 6 Multi-Level TSV Assignment

Require: Grids, netlists
Ensure: TSV assignments for each net η_i

```
1: ε = 0                          // Grid coarsening and source grouping
2: while number of coarsened grids NOT small enough do
3:     while NOT all grids are clustered do
4:         Cluster adjacent k1 × k2 grids to coarsened grd_ij^{+1}
5:     end while
6:     while NOT all source pins are grouped do
7:         Group source pins into group grp_ij^{+1}
8:     end while
9:     ε = ε + 1
10: end while
11: Build highest level of coarsened graph CG
12: while ε > 0 do                    // Graph un-coarsening
13:     Rough flow assignment on graph CG
14:     Generate edges for coarsened graph CG^{-1}
15:     ε = ε − 1
16: end while
17: Detailed TSV assignment on IMCMC network
```

Second, the source pins are grouped, as shown in line 6-8. The source pins that locate in a same coarsened grid are regarded as one group of level ε, denoted as grp_{mn}.

Fig.4.3 shows an example of grid coarsening and source grouping from level 0 to 1. In Fig.4.3(a) the chip dies are divided into small grids. In Fig.4.3(b), the small grids are clustered into larger coarsened grids, for example g_{11} to g_{14} are clustered into grd^1_{11}; then the source pins that locate in a same grid are grouped, for example s_1 and s_2 are grouped into grp^1_{11}, and s_3 and s_4 are grouped into grp^1_{12}.

The grid coarsening is repeated until the number of coarsened grids on each die is small enough, say 2×2.

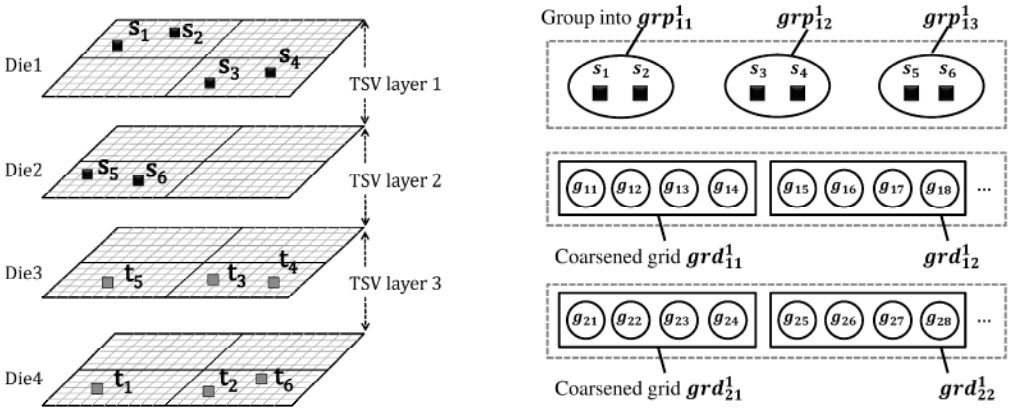
4.4.2 Rough Flow Assignment and Graph Un-coarsening

After grid coarsening and source grouping, rough flow assignment and graph un-coarsening are executed in turns, as shown in line 13 and 14. The rough flow assignment is performed on a directed graph called **coarsened graph**:

Definition 2: A **coarsened graph of level** ε, denoted as $CG = (V_{cg}, E_{cg})$, is a directed graph, whose vertex set is composed by all coarsened grids of level ε and source/sink pins s and t and edge set E_{cg} contains edges between coarsened grids and source groups.

The direction of the edges is the same as the original IMCMC network. Each edge is associated with a cost ω_e, which is the Manhattan Distance between two coarsened grids the distance of adjacent coarsened grids is 1.

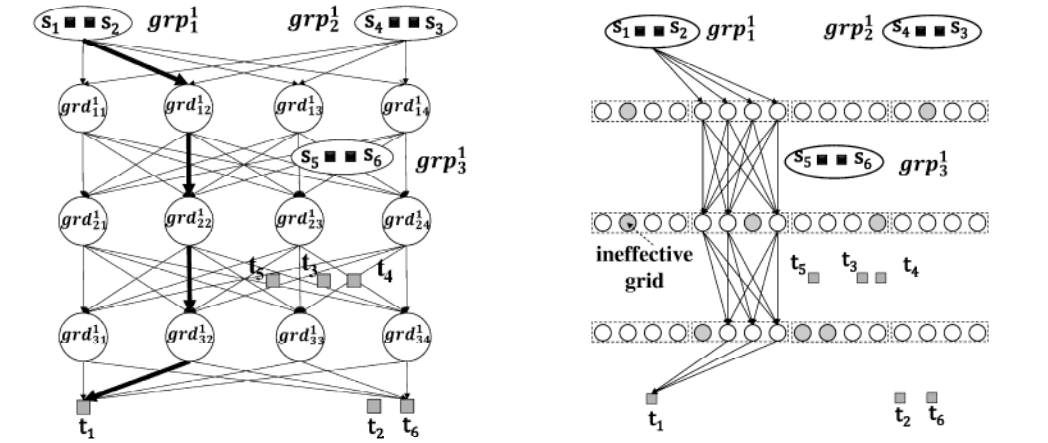
The first coarsened graph of level ε, CG , is a layered graph composed of all coarsened grids



(a) Chip dies and TSV layers are divided into grids

(b) Grid coarsening and source grouping

Figure 4.3: Grids are clustered into coarsened grids, and source pins are grouped according to their locations on chip dies.



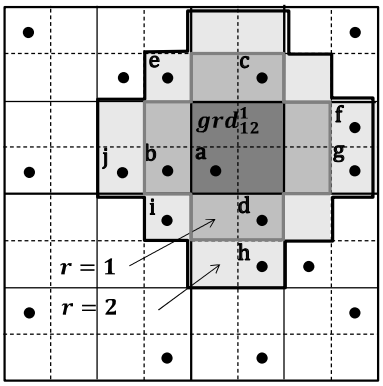
(a) The coarsened graph of level ε with rough flow assignment

(b) Coarsened graph of level $\varepsilon - 1$ with generated edges

Figure 4.4: Rough flow assignment and graph un-coarsening

of level ε and source/sink pins. The edges are generated beforehand between each two coarsened grids of adjacent layers as well as source/sink pins, as shown in line 11 and an example in Fig.4.4(a). On GC , rough flow assignment is first performed, as shown in line 13. Then, based on the flow assignment result on GC , edges are generated for coarsened graph of level $\varepsilon - 1$, CG^{-1}. The grids with non-zero capacities are called *effective grids*; both effective and ineffective grids are included in the coarsened graph, but edges are only generated for effective grids. Fig.4.4(b) shows an example of the coarsened graph of level $\varepsilon - 1$, built based on the flow assignment result from Fig.4.4(a). Edges are only generated for the grids that have flows in graph CG , and ineffective grids (gray ones) are skipped. This procedure repeats till $\varepsilon = 0$.

However, in this problem, a large fraction of grids may be ineffective, and only by splitting grids with flows, the generated edges for the coarsened graph are limited, which may degrade the solution quality.



(a)

Computing the outwards extended grids

Input: left-bottom grid (x_0, y_0)
Output: the set of grids $V = \{(x, y)\}$ to be extended
1 : $r = 0$
2 : **while** (# of grids in V is NOT enough) **do**
3 : $r = r + 1$
4 : **for** $i = 0$ **to** $i < r$ **do**
5 : $V = V + (x_0 + i + 1, y_0 + r - i + 1)$
6 : $V = V + (x_0 + r - i + 1, y_0 - i)$
7 : $V = V + (x_0 - i, y_0 - r + i)$
8 : $V = V + (x_0 - r + i, y_0 + i + 1)$
9 : **end for**
10: **end while**

( )

Figure 4.5: (a) Grid extension on graph of level ε. (b) The procedure to compute the set of grids V to be extended.

Therefore, a **grid extension** is proposed to include more effective grids to generate enough edges. From the coarsened grid grd_{ij} of level ε, denoting the *extending radius* as r, edges are also generated for the effective grids of level $\varepsilon - 1$ which are within the radius r. The extension terminates until enough edges are generated. In additional, for an extended grid, a smaller radius r means it is closer to the grid which has flows from the higher level graph. This radius is also used in a penalty function, which is to be introduced in Section 5.1.2.

Fig.4.5 shows an example of the grid extension. The grid grd_{12}^{1} is split into 4 smaller grids, as shown in Fig.4.5(a); the ones marked with black dots are effective, in this example only grid a is effective. Since the generated edges are insufficient, more grids around a are included. Fig.4.5(a) shows the included grids with $r = 1$ and $r = 2$, and edges are also generated for these grids. Fig.4.5(b) shows the extending procedure to computes the set of grids to be added.

In order not to sacrifice the solution optimality caused by edge reduction, the rough flow assignments are expected to be as optimal as possible. A **mixed single and multi commodity flow algorithm** is proposed to improve the flow assignment optimality, by removing the net order reliance, which is to be introduced in Section 5.

4.5 Rough Flow Assignment: Mixed Single and Multi Commodity Flow

Most previous works, which formulate the TSV assignment as a typical integer multi-commodity flow problem such as[77,76], assign the flows net by net, because each net is regarded as a distinguished commodity. Specifically, shortest paths are found sequentially for each pair of s_i to t_i. As discussed in Section 3, however, assigning flows net by net has a major problem that, once a flow is assigned, it is fixed and cannot be updated; thus the solution optimality greatly relies on the net order.

On the other hand, in this work, given grouped source pins, *if the source pins of some nets belong to a same source group, they can be regarded as one commodity*[80]. For example as shown in Fig.4.6(a), the flow for $\eta_1 = (s_1, t_1)$ and the flow for $\eta_2 = (s_2, t_2)$ are undistinguished: since s_1 and s_2 both belong to group grp_1, the flow from s_1 to t_1 can either be regarded as the one from edges e_1 and e_5, or the one from edges e_2

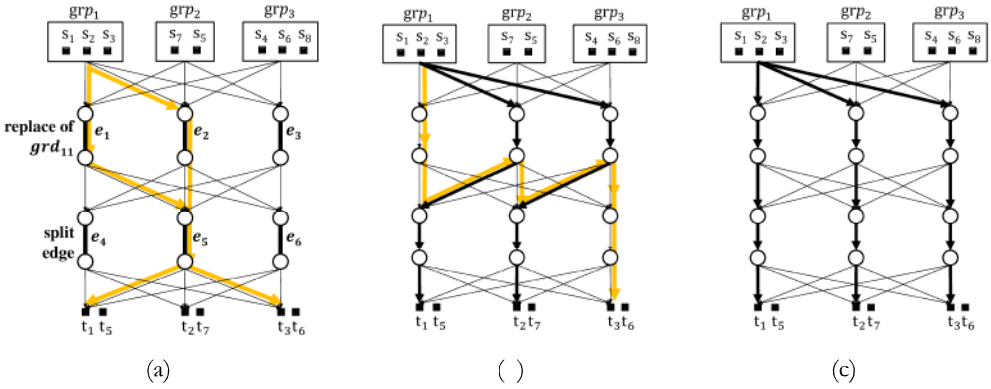


Figure 4.6: (a) Grid vertices are replaced by edges; costs are grid capacities. Flows from a same source group are undistinguished. ( ) Assign a net of grp_1 through the residual edges. (c) Flows of grp_1 are updated.

and e_5. This property allows the single commodity min-cost flow algorithm being adopted for the nets whose source pins are of the same group.

Consequently, benefiting from the coarsened grid and grouped source structure proposed in this work, the multi-commodity and single commodity methods can be mixed by adopting the *successive shortest path* algorithm[78]: when shortest pathes are found net by net, the typical *residual edges*[79] are *on* for the nets belongs to a same commodity, and are *off* for those belong to different commodities.

Now the proposed **mixed multi and single commodity flow algorithm** are explained in detail.

4.5.1 Mixed Single and Multi Commodity Flow Algorithm

Definitions and Preliminaries

Before introducing the mixed flow algorithm, some definitions are first given.

1). Residual edge status and residual network: The same as single commodity min-cost flow, each time a flow f_i is sent, *residual edges*[79] are generated for f_i. However, when sending a new flow f_j from source s_j to sink t_j, where s_j belongs to group grp_{s_j}, not all the residual edges are allowed to load flows; f_j is only allowed to go through residual edges whose flows are of the same commodity, i.e., the sources belong to grp_{s_j}. Therefore, for a flow to be sent from s to t, denoting the source group of s as grp_s, the *status* of a residual edge is defined as:

Definition 3: The **edge status** of a residual edge with respect to grp_s is defined as **on** if the source of the flow on e_r belongs to grp_s, otherwise is **off**.

Accordingly, the residual network is defined as:

Definition 4: The **residual network with respect to grp_s** refers to the network only containing residual edges whose status are on. A shortest augmenting path from s to t can only be found on the residual network for group grp_s.

Fig.4.7 shows examples of edge status and residual network. In Fig.4.7(a), suppose two flows are sent for (s_1, t_1) and (s_2, t_2), denoting f_1 and f_2; two residual edges, e_1 and e_2, are built respectively for f_1 and f_2. In Fig.4.7(b), when a flow is to be sent for (s_3, t_3), where s_3 belongs to group grp_1, the residual

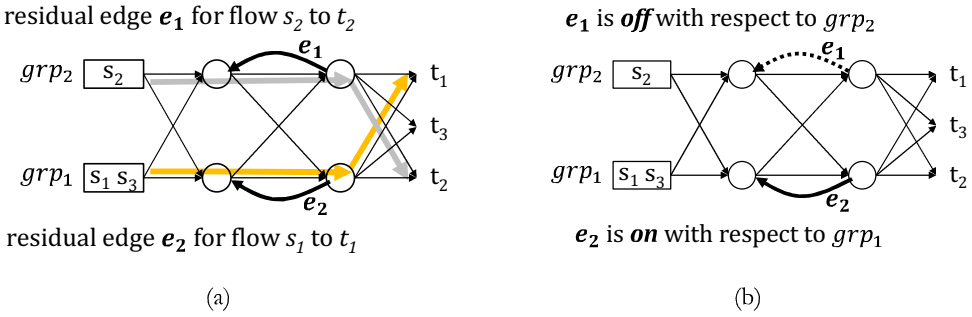


Figure 4.7: Edge status of on and off with respect to source groups. (a) Flows sent for (s_1, t_1) and (s_2, t_2). ($_1$ when sending flow for (s_3, t_3), whose source belongs to grp_1.

edge e_2 with respect to grp_1 is turned on and edge e_1 is turned off. Thus, the residual network with respect to grp_1 contains e_2 but not e_1.

2). Optimally assigned nets: A net is regarded as **optimally assigned** if and only if the grids being assigned to the net satisfy the condition that, the horizontal and vertical coordinates of the grids as well as source and sink pins are monotonous. That is, for a net $\eta = (s, t)$, if the total wire length from s to t equals to the half parameter of the bounding box of η, η is regarded as optimally assigned. For example as shown in Fig.4.8(a), grids g_1, g_3 and g_4 are assigned for Net1 in order; obviously Net1 is not optimally assigned because of the detour from g_3 to g_4. Meanwhile, Net2 is optimally assigned since the wire length from s_2 to t_2 equals to the half parameter of bounding box of Net2.

3). Modified coarsened graph with split edges: Since the capacities of grids, i.e., the number of TSVs can be inserted, is limited, once a grid is fully occupied, other nets may need detours which results in larger wire length or the failure of finding feasible flows. To take grid capacities into consideration to avoid TSV congestions, the coarsened graph is modified: on the coarsened graph, each grid vertex g is replaced by a pair of vertices with a directed edge, called **split edge**, denoted as e_g. The capacity of e_g equals to the capacity of the grid vertex g, i.e., $c_{e_g} = c_g$. Fig.4.6(a) shows an example of the modified coarsened graph, w .

The overall rough flow assignment is shown in Algorithm 2. The successive shortest path based mixed flow method is first applied on groups of source pins, as shown in Line 1-5; then the single commodity min-cost flow algorithm is applied on grid vertices of coarsened graphs, as shown in Line 6-10. The two parts are introduced respectively.

Mixed flow on groups of source pins

In this step, flows are assigned net by net by successive shortest path. The purpose is to first ensure a feasible assignment for each net, and then minimize the total wire length. For each net $\eta = (s, t)$, a residual network for grp_s is built, and a shortest augmenting path p from s to t is found, denoted as $p^{s,t}$.

Algorithm 7 Mixed Single and Multi Commodity Assignment

Require: Coarsened grid graph of level ε
Ensure: Rough flow assignments of all the nets
1: **for** each net $\eta = (s, t)$ **do**
2: $grp_i \leftarrow$ source group of s
3: Build the residual network for group grp_i
4: Find a shortest augmenting path p from s to t, assign the flow for net η on path p
5: **end for**
6: **for** each grid vertex $v \in V_{cg}$ in graph CG **do**
7: **if** NOT all the flows on v are optimized **then**
8: Apply single commodity min-cost flow on vertex v
9: **end if**
10: **end for**

The length of path $p^{s,t}$ is computed as:

$$L(p^{s,t}) = \sum_{e \in p^{s,t}} \omega_e + \sum_{e_g \in p^{s,t}} Pc(c_{e_g}) + \sum_{e_g \in p^{s,t}} Pr(e_g) \tag{4.3}$$

w , indicating the routing cost in terms of wire length to be minimized. $Pc(c_{e_g})$ and $Pr(e_g)$ are *penalty functions*:

$Pc(c_{e_g})$ is a monotonically decreasing function of the capacity of edge e_g. It means the grids with larger capacities are more preferable, in order to avoid TSV congestions.

$Pr(e_g)$ is a monotonically increasing function of the *radius* r (defined in Section 4.2) from grid g (represented by edge e_g) to the grid of 1-level higher which holds the same flow from s to t. This is because on the coarsened graphs of higher levels, especially the highest level, most of the nets are optimally assigned; when assigning flows on lower level graphs, the flows are still expected to be assigned similarly to those of higher levels which are optimally assigned. For example in Fig.4.5(a), suppose a flow going through grid grd^{π}_{12} on the coarsened graph of level $\varepsilon + 1$; the penalties for the grids within radius $r = 1$, say grids b to e, are smaller than the penalties for grids within $r = 2$, say grids f to j.

Fig.4.6(b) and Fig.4.6(c) show examples of the mixed commodity flow algorithm applied on source group of grp_1. In Fig.4.6(b) two flows from grp_1 to t_1 and t_2 are already found. When processing net $\eta_3 = (s_3, t_3)$, because s_3 belongs to the same source group grp_1, an augmenting path is found on the residual network of grp_1, and all the flows of grp_1 are updated as shown in Fig.4.6(c). In this work the Tarjan's algorithm[81] is adopted for shortest path searching.

Min-cost flow on grid vertices

In the stage described in Section 4.5.1, shortest paths are computed including both the wire length and penalty functions; in this stage, an optimization of already assigned flows is applied on grid vertices to

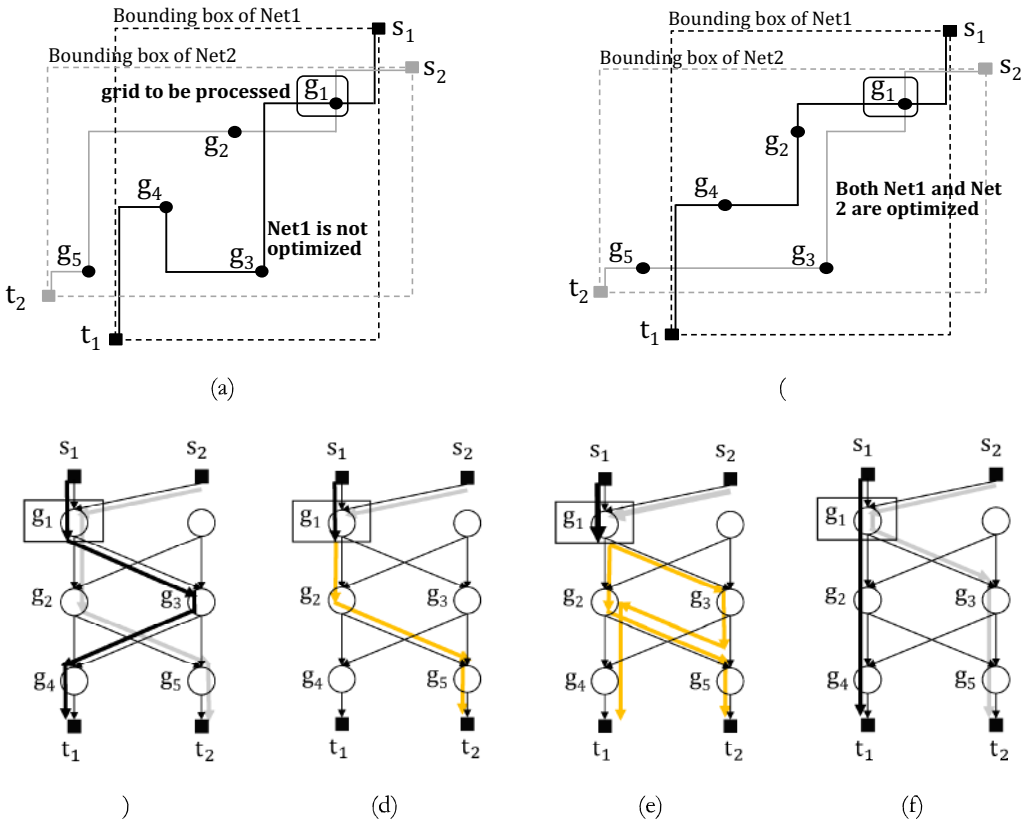


Figure 4.8: The example of single flow algorithm applied on grids. (a) Before processing g_1, Net1 is not optimized. (b) After processing g_1, both Net1 and Net2 are optimized. (c) Initial flow assignment. (d) A shortest path from g_1 to t_2. (e) An augmenting path from g_1 to t_1 going through a residual edge. (f) Updated flow assignments from (c).

minimize total wire length only.

Similar to each source group, for each grid vertex g, all the flows that come into g, or all the flows that come out of g, can also be regarded as a same commodity. Therefore, on each grid vertex g, the min-cost flow algorithm is applicable. For efficiency, only the vertices that *not all nets are optimally assigned* are processed. As shown in line 7 in Algorithm 2, when processing a grid vertex g, it is first examined that whether all the related nets, whose flows go through g, are optimally assigned; if not, the single min-cost flow algorithm is applied on g. Fig.4.8(c) to Fig.4.8(e) show an example of min-cost flow algorithm being applied on grid vertex g_1. Fig.4.8(c) shows the flow assignment of Fig.4.8(a), where Net2 is optimally assigned but Net1 is not. When processing vertex g_1, as shown in Fig.4.8(d), an augmenting path is found from g_1 to t_1 on the residual network. Fig.4.8(e) shows the updated flow assignment of Fig.4.8(b), where both Net1 and Net2 are optimally assigned.

4.6 Extended Layer by Layer TSV Assignment Optimization

In Section 4 and 5 a multi-level algorithm for TSV assignment is proposed, which generates initial solutions. In this section, a TSV assignment optimization is introduced to further reduce total wire length, which is a post-processing.

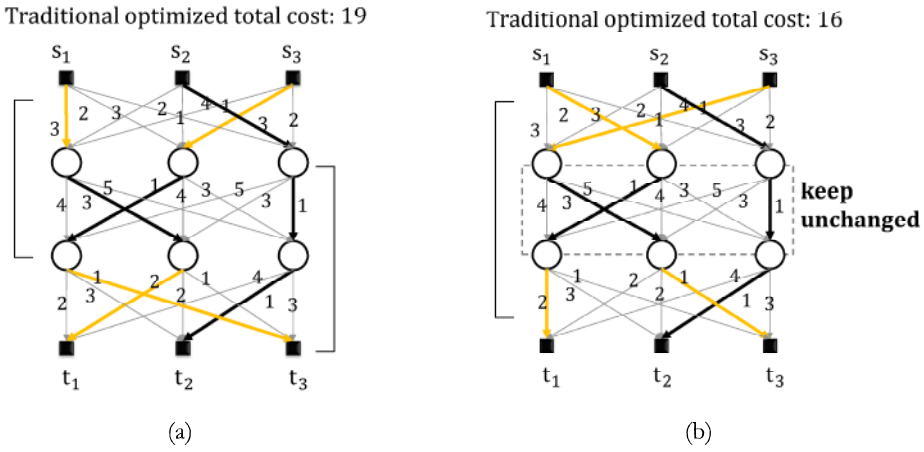


Figure 4.9: (a) Traditional layer by layer optimization with total cost of 19. ( ) Extended layer by layer optimization with total cost of 16.

One existing method is to optimize the TSV assignment layer by layer between two *adjacent chip dies*[76]. This method gets global optimum assignment for adjacent chip dies, but can hardly get global optimum solution when non-adjacent dies are considered. To remove the limitation, in this work, an extended layer by layer optimization is proposed, which is able to optimize any two *non-adjacent chip dies* under the constraint that the flows of other layers are fixed. As shown in Fig.4.9, the left figure is the result after traditional layer by layer optimization, whose total cost is 19, and the right figure shows an improved solution by extended layer by layer optimization, whose total cost is 16.

The optimization is performed on the lowest level of IMCMC network with no coarsened grid anymore. Given an initial flow and two arbitrary TSV layers l_p and l_q, the goal is to optimize the flow assignment for the nets whose pins are on chip dies d_p and d_{q+1}, as shown in Fig.4.10(a). The flows between d_p and d_{q+1} are fixed, as the gray shadow shown in Fig.4.10(a). A flow going out of s_k must go through a fixed flow path and still go into t_k, to guarantee that the reassigned flows are still valid; each fixed flow path has a capacity of 1-unit. For example in Fig.4.10(a), after the extended layer by layer, the flow from s_1 still goes into t_1.

To optimize the flows for chip dies d_p and d_{q+1}, a weighted bipartite graph is built as shown in Fig.4.10(b). The edge weights of the bipartite graph are computed by summing the costs of two original edges on IMCMC network of dies d_p and d_{q+1} which belong to a same flow. For example the weight for edge (s_3t_3, v_3) is $\omega_{19} + \omega_{46}$ because in Fig.4.10(a), ω_{19} and ω_{46} are of the same flow from s_3 to t_3. Then a min-cost matching is found on the bipartite graph and the flow assignment is updated.

The algorithm is executed between every two chip dies d_p and d_{q+1}: when $q - p = 0$, say d_1 and d_2, it is the same as the traditional layer by layer optimization; when $q - p \geq 1$, say d_1 and d_3, it is the extended layer by layer optimization.

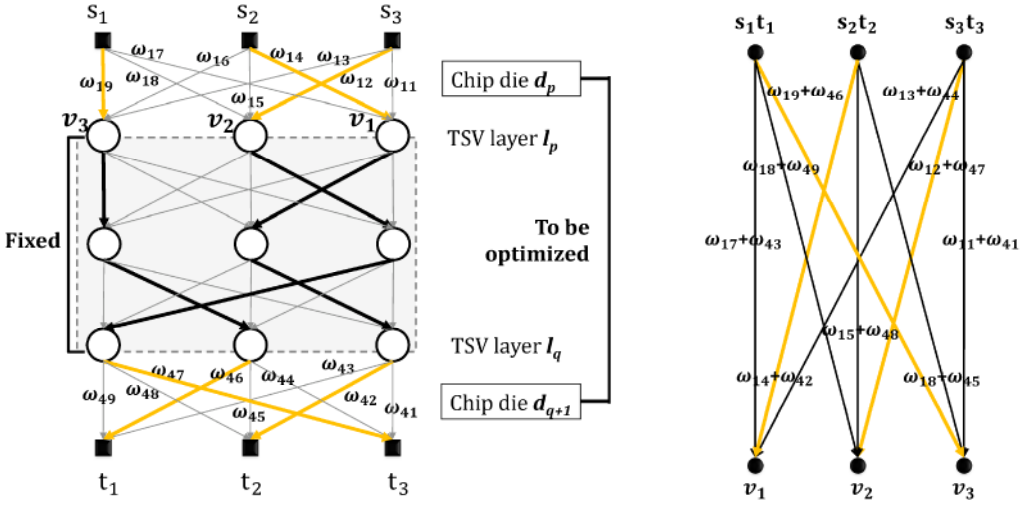


Figure 4.10: (a) Extended layer by layer to optimize the TSV assignments of two non-adjacent dies. (b) The bipartite graph for min-cost matching.

T .1: Edge reduction and algorithm speedup by the multi-level proposal

T Bench	#Nets	#TSVs	Grid Size	Orig. #Edges	Red. #Edges	Cmp	#Edges in *CG*		
							$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 3$
ami33	85	144	62×61	284K	8K	35.5X	3K	2K	1K
ami49	357	652	79×82	1080K	78K	13.8X	26K	20K	9K
n50	382	627	53×53	871K	49K	17.8X	14K	10K	6K
n100	650	1037	52×52	1279K	103K	12.4X	33K	24K	10K
n200	1246	1945	31×31	2166K	187K	11.6X	75K	39K	19K
n300	1489	2325	68×68	5529K	243K	22.8X	94K	82K	29K
AVG						**19.0X**			

4.7 Experimental Results

The experiments are carried on Linux RedHat with 2.8GHz CPU and 8GB memory, implemented by C. The benchmarks come from MCNC[83] and GSRC[84]. All the algorithms are implemented by us on the same platform. The grid size $g_1 \times g_2$ in tables refers to the number of grids on each chip die. The wire length is computed from Eq.4.2. k = = . This work is first compared to the latest existing work[76] in terms of algorithm efficiency and solution quality, and then is compared to previous work[77] in terms of algorithm efficiency. Some other works as[70][72] also discussed the TSV assignment, but they solved it with other problems jointly such as floorplan and placement, which makes it difficult to make a comparison.

Firstly, the edge reductions and algorithm speedups by multi-level approach is shown in Table 1. The Orig.#Edges shows the number of edges without multi-level proposal; the Red.#Edges shows the final reduced number of edges with multi-level proposal, i.e., $\varepsilon = 0$. It shows that on average this

Table 4.2: Comparisons between work[76] and the multi-level proposal with/without penalty function, and with/without min-cost flow on grid vertices.

Test Bench	Chip Layer	Grid Size	**Integ. TSV Assign[76]**		**ML + MF***		**ML + MF + PF***		**ML + MF + PF + FG***	
			Wirelength	Time(ms)	Wirelength	Time	Wirelength	Time	Wirelength	Time
ami33	4	62×61	3235	1478 (44.12)	3186 (0.985)	42	3179 (0.983)	43 (1.024)	3145 (0.972)	46 (1.095)
		75×77	5173	6251 (167.45)	5459 (1.055)	51	5081 (0.982)	51 (1.000)	5073 (0.981)	54 (1.059)
ami49	4	33×33	7153	4277 (16.18)	6705 (0.937)	344	6623 (0.926)	347 (1.009)	6531 (0.913)	369 (1.073)
		41×41	9181	9770 (39.20)	8852 (0.964)	355	8765 (0.953)	358 (1.008)	8483 (0.924)	399 (1.124)
		64×65	12555	11722 (42.65)	12833 (1.022)	408	12159 (0.968)	410 (1.005)	12053 (0.960)	433 (1.061)
		79×82	15480	48503 (204.07)	15345 (0.991)	264	15297 (0.988)	267 (1.011)	14862 (0.960)	367 (1.390)
n50	4	53×53	15580	5700 (24.20)	13459 (0.864)	243	13413 (0.861)	246 (1.012)	13359 (0.857)	362 (1.490)
n100	4	26×26	12193	6787 (7.16)	12003 (0.984)	994	11821 (0.969)	998 (1.004)	11761 (0.965)	1416 (1.425)
		29×29	11345	7754 (7.98)	10331 (0.911)	1093	10203 (0.899)	1100 (1.006)	10181 (0.897)	1149 (1.051)
		33×33	13095	7741 (8.32)	12554 (0.959)	981	12229 (0.934)	994 (1.013)	12129 (0.926)	1051 (1.071)
		38×38	14205	13052 (14.97)	13283 (0.935)	1085	13157 (0.926)	1091 (1.006)	13131 (0.924)	1262 (1.163)
		52×52	22589	15297 (19.88)	21790 (0.965)	806	21600 (0.956)	809 (1.004)	21323 (0.944)	932 (1.156)
n200	4	28×28	22633	45282 (12.00)	21099 (0.932)	4429	20107 (0.888)	4436 (1.002)	20079 (0.887)	5246 (1.184)
		31×31	24840	56861 (14.70)	23176 (0.933)	4712	22358 (0.900)	4727 (1.003)	22346 (0.900)	5137 (1.090)
n300	4	48×48	42757	118182 (28.33)	41857 (0.979)	6960	39823 (0.931)	6981 (1.003)	39787 (0.931)	7581 (1.089)
		56×56	45443	149362 (44.42)	43981 (0.968)	5946	42949 (0.945)	5957 (1.002)	42391 (0.933)	6155 (1.035)
		68×68	57766	150342 (48.97)	56704 (0.982)	3997	55648 (0.963)	4012 (1.004)	54184 (0.938)	4865 (1.217)
AVR			**1.00**	**43.80**	**0.963**	**1.000**	**0.940**	**1.007**	**0.930**	**1.163**

[1] **ML** represents the simple multi-level algorithm, **MF** represents the mixed flow method, **PF** represents penalty functions $Pc(c_{e_g})$ and $Pr(e_g)$ in Eq.3, and **FG** represents the min-cost flow algorithm being applied on grid vertices.

proposal reduces the number of edges by 19X, and speeds up the algorithm by 14.6X. It implies that the unpromising edges are significantly reduced to improve the algorithm efficiency. Also, the number of edges on coarsened graphs of different levels are shown in #Edges in CG . Since the number of levels varies with different testbenches, for simplicity only three levels are shown, i.e., $\varepsilon = 1, 2$, and 3.

In Table 4.2 the proposal is compared to the latest work[76], whose results are shown in Integ. TSV Assign[76]. For this proposal, three sets of solutions are shown: **ML+MF** refers to the multi-level proposal with mixed flow algorithm being applied on group sources, which is the same as my earlier work[64]; **ML+MF+PF** means that the penalty functions $P_c(c_{e_g})$ and $Pr(e_g)$ are included, and **ML+MF+PF+FG** means that the min-cost flow algorithm being applied on the grid vertices, discussed in Section 5.1.3, is also included. First, it shows that the multi-level algorithm with mixed flow proposal reduces the wire length by 3.7% on average compared to[76]. One suggested reason is, in[76], as discussed in Section 3, while computing the shortest path, the possible paths are restricted strictly and the solution quality is sacrificed. On the other hand, in the level by level proposal, most of the flows assigned on higher level graphs are optimal; based on which, only promising edges are generated, so that flows on lower levels are expected to be close to optimal solutions. Then, as

Table 4.3: Evaluation of the traditional layer by layer[77] and the extended layer by layer optimizations.

T Bench	Grid Size	**Traditional LBL[77]**		**Extended LBL**	
		Wirelength	+Time(ms)	Wirelength	+Time(ms)
ami33	62×61	3145 (1.000)	+0 (1.000)	3145 (1.000)	+0 (1.000)
	75×77	5065 (0.998)	+2 (1.043)	5062 (0.998)	+3 (1.058)
ami49	33×33	6531 (1.000)	+4 (1.012)	6502 (0.996)	+6 (1.017)
	41×41	8477 (0.999)	+3 (1.008)	8411 (0.992)	+11 (1.028)
	64×65	12039 (0.999)	+3 (1.008)	12010 (0.996)	+14 (1.034)
	79×82	14854 (0.999)	+4 (1.010)	14828 (0.996)	+22 (1.059)
n50	53×53	13323 (0.997)	+7 (1.018)	13301 (0.996)	+15 (1.040)
n100	26×26	11669 (0.992)	+3 (1.002)	11595 (0.986)	+6 (1.004)
	29×29	10164 (0.998)	+6 (1.005)	10087 (0.991)	+14 (1.012)
	33×33	12122 (0.999)	+12 (1.006)	12009 (0.990)	+17 (1.016)
	38×38	13104 (0.998)	+7 (1.007)	13015 (0.991)	+16 (1.019)
	52×52	21263 (0.997)	+4 (1.003)	21241 (0.996)	+19 (1.027)
n200	28×28	20079 (1.000)	+4 (1.001)	19985 (0.995)	+31 (1.006)
	31×31	22278 (0.997)	+10 (1.002)	22102 (0.989)	+32 (1.006)
n300	48×48	39732 (0.999)	+5 (1.001)	39609 (0.996)	+18 (1.002)
	56×56	42342 (0.999)	+13 (1.002)	42217 (0.996)	+34 (1.007)
	68×68	54130 (0.999)	+7 (1.001)	54008 (0.997)	+34 (1.007)
AVR		**0.998**	**1.008**	**0.994**	**1.020**

shown in ML+MF+PF, including the penalty functions further reduces the wire length by 2.3%, and applying the min-cost flow algorithm on grid vertices reduces another 1.0% as shown in ML+MF+PF+FG. The run time is evaluated by normalizing all the values to the fastest one, ML+MF. The work[76] shows longest execution time, which is 43.8X slower. The time overhead of penalty functions is 0.7%, and the time overhead of min-cost flows on grid vertices is 16.3%. In short, the multi-level proposal runs 37X faster than the previous work and also reduces the wire length by 7.0%.

Besides, the algorithm efficiency is also compared to the previous work[77]. For small testbenches such as *ami*33, *ami*49 and *n*50 with the grid size of 40 × 40, 80 × 80 and 30 × 30, the proposed algorithm runs 6 to 12 times faster than[77]. For larger testbenches such as *n*100 and *n*300, the grids are expected to be more than 30 × 30 since the chip die is larger. For *n*100, the proposed algorithm divides the chip die into 52 × 52, and for *n*300 the grids are 68 × 68. However, due to the efficiency limitation of[77], it failed to produce solutions with grids more than 30 × 30. This implies that the proposed algorithm in this work is much efficient than[77]. As for wire length, this proposed algorithm is also tested for *n*100 and *n*300 with grids of 30 × 30, and the differences of wire length between this proposal and[77] is less than 1%, while the execution time of this proposal is 5 and 7 times faster.

In Table 4.3, the traditional layer by layer (LBL) optimization in[77] and the extended one are evaluated. It shows that the traditional LBL reduces the wire length by 0.2%, and the extended one reduces the wire length by 0.6%. The time consumed by post processing after the multi level

algorithm are also shown. The overheads of the traditional and extended layer by layer post processing are 0.8% and 2%, respectively.

4.8 Summary

In this work an efficient multi-level algorithm for 3D-IC TSV assignment is proposed to minimize the total wire length. It formulates the problem as an integer multiple commodity min cost (IMCMC) flow problem. To reduce the number of edges in the IMCMC network, an iterative multi-level approach is proposed. In each iteration, a coarsened graph is built and the problem is roughly solved on the coarsened graph, then a next-level coarsened graph is generated based on the rough solution to be solved in the next iteration. Benefitting from the multi-level proposal, and in order to not to degrade solution quality, a mixed single and multi-commodity min-cost flow method is proposed for solution optimization. Besides, an extend layer by layer optimization method is also discussed. The experimental results show that the proposed multi-level proposal achieves 37X speedup compared to the previous work, and meanwhile reduces the total wire length by 7.0%. The extended layer by layer optimization further reduces 0.6% total wire length.

5 Conclusions and Future Work

5.1 Conclusions

In this thesis, the following problems in the EDA design flow are discussed and solved using common technologies, graph theory, mathematical programming and iterative approaches.

1. **Improved the solution optimality for small and medium scaled problems in HLS steps**, including: (1) multiple voltage scheduling (combined with binding) problem for power minimization, and (2) port assignment problem for interconnection complexity reduction. For (1), a mathematical formulation, the ILP formulation, is first given, and by relaxing ILP to LP, the network simplex method in graph theory is adopted for speedup. For (2), a graph theory based tree transformation approach is first proposed, and by substituting $\oplus$ for $+$ in LP, the LP pivotings in mathematical programming is adopted for speedup. Besides, for both (1) and (2), a local search with random restart based iterative method is proposed for solution optimization. In the experiments, optimal or near-optimal solutions for both problem (1) and (2) are obtained. For (1), the dynamic power minimization solutions are optimum compared to ILP, and the leakage power minimization solutions saves 43.9% power compared to latest existing work, with a 52 times speedup; For (2), the proposed algorithm gets optimum solutions with a probability of more than 99% with less than 4% execution time overhead.

2. **Improved the algorithm efficiency for a large scaled problem in physical synthesis: the TSV insertion problem for 3D-ICs**. Given huge size inputs, an iterative multi-level approach is proposed. In each iteration, first a coarsened graph is built and the problem is roughly solved on the coarsened graph; the rough solutions are used to generate a coarsened graph in the next iteration to be solved. Benefitting from the multi-level proposal, and in order to not to degrade solution quality, a graph theory based mixed single and multi-commodity min-cost flow method is proposed for solution optimization, followed by a bipartite matching optimization. In the experiments, the large scaled problems are much more efficiently solved by the proposed

algorithms compared to existing algorithms, where 7% total wire length is reduced with 37 times speedup.

To briefly conclude, in this thesis, it shows that the graph theory, mathematical programming, as well as the iterative methods, are effective technologies in solving EDA design problems, for example improving solution quality for small and medium scaled problems, and improving algorithm efficiency for large scaled problems.

5.2 Open Issues and Future Work

Although some achievements are made in this thesis, there are still several open issues on each topic:

1. **Dynamic and leakage power minimization**[3]: (a) In this work, the proposed unified algorithm is conditionally applicable — when the possible delays (number of control steps) of operations are consecutive integers, and when the power-delay function curve is convex. In this work the data of FU delays and power-delay functions are taken from previous related works, where the conditions are satisfied; but it is possible that the conditions are not satisfied. (b) for the dynamic and leakage power co-optimization problem in this work, there is an assumption that, given both multiple V_{dd} and multiple V_{th}, the delays of FUs under different pairs of V_{dd} and V_{th} are distinguished. However this assumption might be limited.

2. **Interconnection allocation**[35]: In this work two optimizations, the tree based and matrix based, are discussed, while in my previous works the bipartite graph based optimization have also been studied. It shows that the bipartite graph based algorithm runs faster than both tree based and matrix based, but the solution quality is lower. This is because in each iteration, the number of candidate neighborhood solutions to be searched are less. Even though, it is still possible that a bipartite graph approach excels the matrix approach in both solution quality and efficiency.

3. **Through silicon via (TSV) insertion for 3D-IC**[71]: In the multi-level approach of this work, in each iteration, the coarsened graph of the current level is generated according to the rough solution, i.e., the flow assignment results on the coarsened graph of the previous level. Currently, after generating the coarsened graph, the previous flow assignment results are abandoned, and the flow assignment on the new generated graph is computed independently. However, since the coarsened graphs of higher levels are small, the flow assignment results on them are closer to optimum solutions. Therefore, it is suggested that the flow assignment results on the higher level coarsened graphs may be used to guide the flow assignment on the lower level graphs, to further improve the solution quality.

Given the open issues, there are future works left for further investigation:

1. In the method of multiple supply/threshold voltage scheduling, first, when the possible delays of operations are not consecutive integers, or the power-delay function is not convex, for leakage and dynamic power minimization, when multiple V_{dd} and multiple V_{th} voltages are given at the same time, additional heuristic processing will be needed. Besides, one operation delay may

correspond to more than one pairs of V_{dd} and V_{th}. In this case, a heuristic method is needed to select from different pairs of V_{dd} and V_{th}. Also, when the possible delays of an FU are not consecutive integers, say $(1, 2, 4)$, and the FU is assigned by delay 3, another processing, for example dynamic programming or branch and bound, is needed to re-assign the delay of the FU to a valid value. Finally, the pipeline operators and FU chaining are also expected to be considered.

2. In the method of interconnection optimization, there are several improvements for current algorithms that can be considered. For example, first, try to adopt tabu search in the iterative method instead the current local search. Second, in the current algorithm, computing the minimum vertex cover from the conflict graph in each iteration consumes a large fraction of execution time; therefore, it is suggested that if the minimum vertex cover size can be estimated from the conflict graph without actually computing, the CPU time of the current algorithm may be shortened. Besides, it is expected that if FU and register binding are not fixed and being solved with port assignment jointly, and the number of interconnections can be further reduced. Finally, the port assignment problem is expected to be solved on chained FUs with more than two input ports, say three, and more explorations are expected.

3. In the method of 3D-IC TSV insertion, to improve the quality of flow assignment on coarsened graphs by utilizing the flow assignment result on the previous level graph can be considered. Besides, the chip thermal and hotshots are expected to be considered jointly with the wire length.

References

[1] Gajski, Daniel D., Nikil D. Dutt, Allen CH Wu, and Steve YL Lin, "High-Level Synthesis: Introduction to Chip and System Design", Springer Science & Business Media, 2012.

[., Gajski, D.D., Meredith, M. and Takach, A., "An introduction to high-level synthesis", IEEE Design & Test of Computers, 26(4), pp. 8-17, 2009.

[3] Hao, C., Wang, N. and Yoshimura, T., "A Unified Scheduling Approach for Power and Resource Optimization with Multiple V-dd or/and V-th in High Level Synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1-14, Jan. 2017.

[., Chen, S. and Yoshimura, T., "Network simplex method based Multiple Voltage Scheduling in Power-efficient High-level synthesis", IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 237-242, Jan. 2013.

[5] Zhang, Z., Chen, D., Dai, S. and Campbell, K., "High-level synthesis for low-power design", IPSJ Transactions on System LSI Design Methodology 8, pp. 12-25, 2015.

[6] Raghunathan, A., Jha, N.K. and Dey, S., "High-level power analysis and optimization", Springer Science & Business Media, 2012.

[7] Khouri, K. S. and Jha, N. K., "Leakage power analysis and reduction during behavioral synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 10(6), pp. 876-885, 2002.

[8] Tang, X., Zhou, H. and Banerjee, P., "Leakage power optimization with dual-V_{th} library in high-level synthesis", In Proceedings of the 42nd annual Design Automation Conference, pp. 202-207, Jun. 2005.

[9] Chen, Y., Xie, Y., Wang, Y. and Takach, A., "Minimizing leakage power in aging-bounded high-level synthesis with design time multi-V th assignment", IEEE Asia and South Pacific In Design Automation Conference (ASP-DAC), pp. 689-694, Jan. 2010.

[10] Chen, D., Cong, J., Fan, Y. and Xu, J., "Optimality study of resource binding with multi-Vdds", In Proceedings of the 43rd annual Design Automation Conference, pp. 580-585, Jul. 2006.

[11] Gu, Z. P., Yang, Y., Wang, J., Dick, R. P. and Shang, L., "TAPHS: Thermal-aware unified physical-level and high-level synthesis", In Proceedings of the 2006 Asia and South Pacific Design Automation Conference, pp. 879-885, Jan. 2006.

[12] Liu, H. Y., Lee, W. P. and Chang, Y. W., "A provably good approximation algorithm for power optimization using multiple supply voltages", In Proceedings of the 44th Annual Design Automation Conference, pp. 887-890, Jun. 2007.

[13] Jiang, W., Zhang, Z., Potkonjak, M. and Cong, J., "Scheduling with integer time budgeting for low-power optimization", In Proceedings of the 2008 Asia and South Pacific Design Automation Conference, pp. 22-27, Jan. 2008.

[14] Shin, I., Paik, S., Shin, D. and Shin, Y., "Hls-d -level synthesis framework for dual-vdd architectures", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20(4), pp. 593-604, 2012.

[15] Dhillon, Y. S., Diril, A. U., Chatterjee, A. and Lee, H. H. S., "Algorithm for achieving minimum energy consumption in CMOS circuits using multiple supply and threshold voltages at the module level", In Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design, p. 693, Nov. 2003.

[16] Srivastava, A. and Sylvester, D., "Minimizing total power by simultaneous V_{dd}/V_{th} assignment", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23(5), pp. 665-677, 2004.

[17] Nomura, M., Ikenaga, Y., Takeda, K., Nakazawa, Y., Aimoto, Y. and Hagihara, Y., "Delay and power monitoring schemes for minimizing power consumption by means of supply and threshold voltage control in active and standby modes", IEEE Journal of solid-state circuits, 41(4), pp.805-814, 2006.

[18] Haghdad, K. and Anis, M., "Design-specific optimization considering supply and threshold voltage variations", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(10), pp. 1891-1901, 2008.

[19] Brualdi, R. A. and Ryser, H. J., "Combinatorial matrix theory", Cambridge University Press, Vol.39, 1991.

[. A. and Camposano, R. eds., "A survey of high-level synthesis systems", Springer Science & Business Media, Vol.135, 2012.

[21] Lee, C., Potkonjak, M. and Mangione-Smith, W. H., "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems", In Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, pp. 330-335, Dec. 1997.

[., Song, C., Zhong, W., Nan, L. and Yoshimura, T., "Mobility Overlap-Removal-Based Leakage Power and Register-Aware Scheduling in High-Level Synthesis", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 97(8), pp. 1709-1719, 2014.

[., Zhong, W., Hao, C., Chen, S., Yoshimura, T. and Zhu, Y., "Leakage-Power-Aware Scheduling With Dual-Threshold Voltage Design", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(10), pp. 3067-3079, 2016.

[24] Lin, T., Dong, S., Chen, S., Ma, Y., He, O. and Goto, S., "Novel and efficient min cut based voltage assignment in gate level", In Proceedings of the 12th International Symposium on Quality Electronic Design, pp. 1-6, Mar. 2011.

[25] Chen, H. I., Loo, E. K., Kuo, J.B. and Syrzycki, M. J., "Triple-threshold static power minimization in high-level synthesis of VLSI CMOS", In International Workshop on Power and Timing Modeling, Optimization and Simulation, Springer Berlin Heidelberg, pp. 453-462, Sep. 2007.

[. and Lohani, H., "Design, implementation and performance comparison of multiplier topologies in power-delay space", Engineering Science and Technology, an International Journal, 19(1), pp. 355-363, 2016.

[., Ni, J. M., Wang, H. T. and Yoshimura, T., "Simultaneous scheduling and binding for resource usage and interconnect complexity reduction in high-level synthesis", In Proceedings of the 11th International Conference on ASIC (ASICON), pp. 1-4, Nov. 2015.

[. and Vishal M., "Design and Comparison of Multiplexer using Different Methodologies", International Journal of Advanced Research in Computer and Communication Engineering, Vol.5, Issue 4, pp. 478-480, Apr. 2016.

[., Wu, W., Yang, J., Zhang, C. and Zhang, Y., "Reduce register files leakage through discharging cells", In Proceedings of the International Conference on Computer Design, pp. 114-119, Oct. 2006.

[., Avya, L. J. and McCluskey, E. J., "Efficient multiplexer synthesis techniques", IEEE Design & Test of Computers, 17(4), pp. 90-97, 2000.

[31] Paulin, P. G. and Knight, J. P., "Force-directed scheduling for the behavioral synthesis of ASICs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 8(6), pp. 661-679, 1989.

[. and Liu, X., "Compatibility path based binding algorithm for interconnect reduction in high level synthesis", In Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design, pp. 435-441, Nov. 2007.

[., Chen, S. and Yoshimura, T., "Port assignment for interconnect reduction in high-level synthesis", In Proceedings of the International Symposium on VLSI Design, Automation, and Test (VLSI-DAT), pp. 1-4, Apr. 2012.

[., Zhang, H. R., Chen, S., Yoshimura, T. and Wu, M. Y., "Port assignment for multiplexer and interconnection optimization", In Proceedings of the 5th Asia Symposium on Quality Electronic Design, pp. 136-143, Aug. 2013.

[., Ni, J., Wang, N. and Yoshimura, T., "Interconnection allocation between functional units and registers in high-level synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(3), pp. 1140-1153, 2017.

[., Wang, N., Chen, S., Yoshimura, T. and Wu, M. Y., "Interconnection allocation between functional units and registers in High-Level Synthesis", In Proceedings of the 10th International Conference on ASIC, pp. 1-4, Oct. 2013.

[. Svensson, "Power consumption estimation in CMOS VLSI chips", In IEEE Journal of Solid-State Circuits, vol. 29, no. 6, pp. 663-670, Jun. 1994.

[38] Magen, N., Kolodny, A., Weiser, U. and Shamir, N., "Interconnect-power dissipation in a microprocessor", In Proceedings of the 2004 international workshop on System level interconnect prediction, pp. 7-13, Feb. 2004.

[., Cong, J., Fan, Y. and Wan, L., "LOPASS: A low-power architectural synthesis system for FPGAs with interconnect estimation and optimization", IEEE transactions on very large scale integration (VLSI) systems, 18(4), pp. 564-577, 2010.

[40] Chen, D., Cong, J., Fan, Y. and Zhang, Z., "High-level power estimation and low-power design space exploration for FPGAs", In Proceedings of the 2007 Asia and South Pacific Design Automation Conference, pp. 529-534, Jan. 2007.

[41] Hsieh, C. T., Cong, J., Zhang, Z. and Chang, S. C., "Behavioral synthesis with activating unused flip-flops for reducing glitch power in FPGA", In Proceedings of the 2008 Asia and South Pacific Design Automation Conference, pp. 10-15, Jan. 2008.

[42] Ahuja, S., Lakshminarayana, A. and Shukla, S. K., "Power reduction using high-level clock-gating", In Low Power Design with High-Level Power Estimation and Power-Aware Synthesis, Springer New York, pp. 119-129, 2012.

[43] Kim, T. and Liu, X., "A global interconnect reduction technique during high level synthesis", In Proceedings of the 15th Asia and South Pacific Design Automation Conference, pp. 695-700, Jan. 2010.

[44] Chen, Y. G., Shi, Y., Lai, K. Y., Hui, G. and Chang, S. C., "Efficient multiple-bit retention register assignment for power gated design: Concept and algorithms", In Proceedings of the International Conference on Computer-Aided Design, pp. 309-316, Nov. 2012.

[45] Choi, E., Shin, C., Kim, T. and Shin, Y., "Power-gating-aware high-level synthesis", In Proceedings of the 2008 international symposium on Low Power Electronics & Design, pp. 39-44, Aug. 2008.

[46] Cortadella, J., Galceran-Oms, M., Kishinevsky, M. and Sapatnekar, S. S., "RTL synthesis: from logic synthesis to automatic pipelining", Proceedings of the IEEE, 103(11), pp. 2061-2075, 2015.

[47] https://embedded.eecs.berkeley.edu/Alumni/rajeev/cs252/report/main/node11.html

[48] Manzak, A. and Chakrabarti, C., "Variable voltage task scheduling algorithms for minimizing energy/power", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 11(2), pp. 270-276, 2003.

[49] Jha, N. K., "Low power system scheduling and synthesis", In Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, pp. 259-263, Nov. 2001.

[50] Chang, H. and Sapatnekar, S. S., "Full-chip analysis of leakage power under process variations, including spatial correlations", In Proceedings of the 42nd annual Design Automation Conference, pp. 523-528, Jun. 2005.

[51] Pangrle, B. M., "On the complexity of connectivity binding", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10(11), pp. 1460-1465, 1991.

[52] Raje, S. and Bergamaschi, R. A., "Generalized resource sharing", In Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design, pp. 326-332, Nov. 1997.

[53] Yamada, S., "An optimal block terminal assignment algorithm for vlsi data path allocation", IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, 80(3), pp. 564-566, 1997.

[54] Chen, D. and Cong, J., "R

. 68-73, Jan. 2004.

[55] Brisk, P. and Ienne, P., "On the complexity of the port assignment problem for binary commutative operators in high-level synthesis", In Proceedings of the International Symposium on VLSI Design, Automation and Test, pp. 339-342, Apr. 2009.

[56] LP-Solver, http://lpsolve.sourceforge.net/5.5/

[57] Kasyanov, V. N. and Evstigneev, V. A., "Graph theory for programmers: algorithms for processing trees", Vol. 515, Springer Science & Business Media, 2000.

[58] Foulds, Leslie R., "Graph theory applications", Springer Science & Business Media, 2012.

[59] Bunch, J. R. and Rose, D. J. eds., "Sparse matrix computations", Academic Press, 2014.

[60] Kasana, H. S. and Kumar, K. D., "Introductory operations research: theory and applications", Springer Science & Business Media, 2013.

[61] Papadimitriou, C. H. and Steiglitz, K., "Combinatorial optimization: algorithms and complexity", Courier Corporation, 1982.

[62] TGFF: avaiable at http://ziyang.eecs.umich.edu/ dickrp/tgff/

[63] Balasubramanian, P. and Edwards, D. A., "Power, delay and area efficient self-t

. 173-178, Apr. 2009.

[64] Hao, C., Ding, N. and Yoshimura, T., "An efficient algorithm for 3D-IC TSV assignment", In Proceedings of the 14th IEEE International New Circuits and Systems Conference, pp. 1-4, Jun. 2016.

[65] Cong, J., Luo, G., Wei, J. and Zhang, Y., "Thermal-aware 3D IC placement via transformation", In Proceedings of Asia and South Pacific Design Automation Conference, pp. 780-785, Jan. 2007.

[66] Fischbach, R., Knechtel, J. and Lienig, J., "Utilizing 2D and 3D rectilinear blocks for efficient IP reuse and floorplanning of 3D-integrated systems", In Proceedings of the 2013 ACM international symposium on International symposium on physical design, pp. 11-16, Mar. 2013.

[67] Yan, H., Li, Z., Zhou, Q. and Hong, X., "Via assignment algorithm for hierarchical 3D placement", In Proceedings of International Conference on Communications, Circuits and Systems, pp. 1229, May 2005.

[68] Cong, J. and Zhang, Y., "Thermal-driven multilevel routing for 3-D ICs", In Proceedings of the 2005 Asia and South Pacific Design Automation Conference, pp. 121-126, Jan. 2005.

[69] Pavlidis, V. F. and Friedman, E. G., "Via placement for minimum interconnect delay in three-dimensional (3D) circuits", In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, pp. 4-pp, May 2006.

[70] Tsai, M. C., Wang, T. C. and Hwang, T., "Through-silicon via planning in 3-D floorplanning", IEEE transactions on very large scale integration (vlsi) systems, 19(8), pp. 1448-1457, 2011.

[71] Cong, H. and Yoshimura, T., "An Efficient Multi-Level Algorithm for 3D-IC TSV Assignment", IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, 100(3), pp. 776-784, 2017.

[72] Kim, D.H., Athikulwongse, K. and Lim, S.K., "Study of through-silicon-via impact on the 3-D stacked IC layout", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 21(5), pp. 862-874, 2013.

[73] Shi, B., Serafy, C. and Srivastava, A., "Co-optimization of TSV assignment and micro-channel placement for 3D-ICs", In Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI, pp. 337-338, May 2013.

[74] Hsu, M. K., Balabanov, V. and Chang, Y. W., "TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 32(4), pp. 497-509, 2013.

[75] Li, C. R., Mak, W. K. and Wang, T. C., "Fast fixed-outline 3-D IC floorplanning with TSV co-placement", IEEE transactions on very large scale integration (VLSI) systems, 21(3), pp. 523-532, 2013.

[76] Liu, X., Yeap, G., Tao, J. and Zeng, X., "Integrated algorithm for 3-D IC through-silicon via assignment", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(3), pp. 655-666, 2014.

[77] Song, C., Liangwei, G., Chiang, M. F. and Yoshimura, T., "Lagrangian relaxation based inter-layer signal via assignment for 3-D ICs", IEICE transactions on fundamentals of electronics, communications and computer sciences, 92(4), pp. 1080-1087, 2009.

[78] Edmonds, J. and Karp, R. M., "Theoretical improvements in algorithmic efficiency for network flow problems", Journal of the ACM (JACM), 19(2), pp. 248-264, 1972.

[79] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B., "Network flows: theory, algorithms, and applications", 1993.

[80] Matveenko, V. D., "Some multicommodity transport problems reducible to a single-commodity problem", USSR Computational Mathematics and Mathematical Physics, 22(1), pp. 70-80, 1982.

[81] Tarjan, R. E., "Data structures and network algorithms", Society for industrial and Applied Mathematics, 1983.

[82] Cormen T. H., "Introduction to algorithms", MIT press, 2009.

[83] MCNC benchmarks. (http://lyle.smu.edu/manikas/Benchmarks/MCNC_Benchmark_Netlists.html)

[84] Gsrc floorplan benchmarks (http://vlsicad.eecs.umich.edu/BK/GSRCbench/)

A

Publication List

Journal Paper

1. Cong Hao, Nan Wang, Takeshi Yoshimura, "A Unified Scheduling Approach for Power and Resource Optimization with Multiple V-dd or/and V-th in High Level Synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) Systems*, pp. 1-14, Jan. 2017.

2. Cong Hao, Takeshi Yoshimura, "An Efficient Multi-Level Algorithm for 3D-IC TSV Assignment", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100-A, No. 3, pp. 776-784, Mar. 2017.

3. Cong Hao, Jianmo Ni, Nan Wang, Takeshi Yoshimura, "Interconnection Allocation Between Functional Units and Registers in High-Level Synthesis", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, No. 99, pp. 1-14, Sep. 2016.

4. Nan Wang, Wei Zhong, Cong Hao, Song Chen, Takeshi Yoshimura, Yu Zhu, "Leakage-Power-Aware Scheduling With Dual-Threshold Voltage Design", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3067-3079, Mar. 2016.

5. Nan Wang, Song Chen, Cong Hao, Haoran Zhang, Takeshi Yoshimura, "Leakage Power Aware Scheduling in High-Level Synthesis", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E97-A, no. 4, pp. 940-951, Apr. 2014.

International Conference Paper (with review)

6. Cong Hao, Takeshi Yoshimura, "Economical Smart Home Scheduling for Single and Multiple Users", *In 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1-4, Oct. 2016.

7. Hui Zhu, Cong Hao, Takeshi Yoshimura, "Thermal-Aware Floorplanning for NoC-Sprinting", *In 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1-4, Oct. 2016.

8. Jiayi Ma, Cong Hao, Takeshi Yoshimura, "Power-efficient Partitioning and Cluster Generation Design for Application-Specific Network-on-Chip", *In 2016 13th International Soc Design Conference (ISOCC)*, pp. 1-4, Oct. 2016.

9. Cong Hao, Nan Ding, Takeshi Yoshimura, "An Efficient Algorithm for 3D-IC TSV Assignment", *In 2016 14th IEEE International NEWCAS Conference (NEWCAS)*, pp. 1-4, Jun. 2016.

10. Cong Hao, Jianmo Ni, Hui-Tong Wang, Takeshi Yoshimura, "Simultaneous Scheduling and Binding For Resource Usage and Interconnect Complexity Reduction in High-Level Synthesis", *In 2015 IEEE 11th International Conference on ASIC (ASICON)*, pp. 1-4, Oct. 2015.

11. Jianmo Ni, Qian Ai, Cong Hao, Takeshi Yoshimura, Nan Wang, "Primal-Dual Method based Simultaneous Functional Unit and Register Binding", *In 2015 IEEE 10th International Conference on ASIC (ASICON)*, pp. 1-4, Oct. 2015.

12. Cong Hao, Nan Wang, Song Chen, Takeshi Yoshimura, Min-You Wu, "Interconnection allocation between functional units and registers in High-Level Synthesis", *In 2013 IEEE 10th International Conference on ASIC (ASICON)*, pp. 1-4, Oct. 2013.

13. Nan Wang, Cong Hao, Nan Liu, Haoran Zhang, Takeshi Yoshimura, "Timing and resource constrained leakage power aware scheduling in high-level synthesis", *In 2013 IEEE 10th International Conference on ASIC (ASICON)*, pp. 1-4, Oct. 2013.

14. Haoran Zhang, Cong Hao, Nan Wang, Song Chen, Takeshi Yoshimura, "Power and resource aware scheduling with multiple voltages", *In 2013 IEEE 10th International Conference on ASIC (ASICON)*, pp. 1-4, Oct. 2013.

15. Cong Hao, Hao-Ran Zhang, Song Chen, Takeshi Yoshimura, Min-You Wu, "Port assignment for multiplexer and interconnection optimization", *In 2013 5th Asia Symposium on Quality Electronic Design (ASQED)*, pp. 136-143, Sep. 2013.

16. Cong Hao, Song Chen, Takeshi Yoshimura, "Network simplex method based Multiple Voltage Scheduling in Power-efficient High-level synthesis", *In 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC))*, pp. 237-242, Feb. 2013.

17. Cong Hao, Song Chen, Takeshi Yoshimura, "Port assignment for interconnect reduction in high-level synthesis", *In 2012 International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*, pp. 1-4, Apr. 2012.

International Workshop (with review)

18. Cong Hao, Takeshi Yoshimura, "EACH: An Energy-Efficient High-Level Synthesis Framework for Approximate Computing", *In 2016 2nd Workshop On Approximate Computing (WAPCO)*, Jan. 2016.

19. Cong Hao, Nan Wang, Jianmo Ni, Takeshi Yoshimura, "An Efficient Tabu Search Methodology for Port Assignment Problem in High-Level Synthesis", *In 2015 24th International Workshop on Logic & Synthesis (IWLS)*, Jun. 2015.

Award

1. Cong Hao, Nan Ding, Takeshi Yoshimura., An Efficient Algorithm for 3D-IC TSV Assignment, In 2016 14th IEEE NEWCAS, *Best Student Paper*

2. Cong Hao, Jian-Mo Ni, Hui-Tong Wang, and Takeshi Yoshimura., Simultaneous Scheduling and Binding For Resource Usage and Interconnect Complexity Reduction in High-Level Synthesis, In 2015 IEEE ASICON, *Best Student Paper*

3. Cong Hao, Song Chen, and Takeshi Yoshimura., Network simplex method based Multiple Voltage Scheduling in Power-efficient High-level synthesis. In 2013 IEEE ASP-DAC, *IEICE VLD Excellent Student Award*

4. Cong Hao, Nan Wang, Song Chen, Takeshi Yoshimura, and Min-You Wu., Interconnection allocation between functional units and registers in High-Level Synthesis, In 2013 IEEE ASICON, *Best Student Paper*

5. Cong Hao, Song Chen, and Takeshi Yoshimura., Port assignment for interconnect reduction in high-level synthesis. In 2012 IEEE VLSI-DAT, *Best Paper Candidate*