UNIVERSITY OF CALIFORNIA, SAN DIEGO

Physical Planning to Embrace Interconnect Dominance in Power and Performance

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Renshen Wang

Committee in charge:

Professor Chung-Kuan Cheng, Chair
Professor Peter Asbeck
Professor Fan Chung Graham
Professor Ronald Graham
Professor Russell Impagliazzo
Professor Jeffrey Remmel

2010

Copyright
Renshen Wang, 2010
All rights reserved.

The dissertation of Renshen Wang is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Professor Chung-Kuan Cheng, who gave me the opportunity to study in UCSD, and then constantly gives me invaluable advices, criticisms and encouragements. His ideas and insights on research topics, problems and methodologies are the critical factor of all our accomplishments. It is a great honor to work with and learn from him.

I would also like to thank my dissertation committee members, who in alphabetical order are, Professor Peter Asbeck, Professor Fan Chung Graham, Professor Ronald Graham, Professor Russell Impagliazzo, Professor Jeffrey Remmel. Their insightful reviews greatly help me improve my dissertation. I am also lucky to have worked in a lab beside Professor Fan Graham and Professor Ronald Graham's offices. From them I received a lot of intelligent and helpful advices.

It is a great pleasure to be a member of VLSI/CAD group, and to be a member of CSE, UCSD. During these years, I live in a friendly and enjoyable environment, being able to work with many talented colleagues. Also I received help by discussions and advices from Professor Evangeline Young in the Chinese Univeristy of Hong Kong and Professor Andrew Kahng in our department.

Finally, I need to thank my parents for years of nurturing and supporting. I am proud of them as they are proud of me. This dissertation, with all my effort and endeavor, is the best present I can give to them.

Chapter 2, in part, is a reprint of the paper "Complexity of 3-D Floorplans by Analysis on Graph Cuboidal Dual Hardness", co-authored with Evangeline Young and Chung-Kuan Cheng. To appear in *ACM Transactions on Design Automation of Electronic Systems*. The dissertation author is the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the paper "Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications", co-authored with Yulei Zhang, Nan-Chi Chou, Evangeline Young, Chung-Kuan Cheng, Ronald Graham. Invited submission to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. The dissertation author is the primary investigator and author of this paper.

VITA

2005	B. S. in Computer Science, Tsinghua University, Beijing
2007	M. S. in Computer Science, University of California, San Diego
2005-2010	Graduate research assistant and teaching assistant, University of California, San Diego
2010	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

Renshen Wang, Evangeline Young, Chung-Kuan Cheng. "Complexity of 3-D Floorplans by Analysis on Graph Cuboidal Dual Hardness". To appear in *ACM Transactions on Design Automation of Electronic Systems*.

Renshen Wang, Yulei Zhang, Nan-Chi Chou, Evangeline Young, Chung-Kuan Cheng, Ronald Graham. "Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications". Invited submission to *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*.

Renshen Wang, Evangeline F. Y. Young, Ronald Graham, Chung-Kuan Cheng. "Physical Synthesis of Bus Matrix for High Bandwidth Low Power On-chip Communications" *ACM International Symposium on Physical Design*, 2010.

Renshen Wang, Nan-Chi Chou, Bill Salefski and Chung-Kuan Cheng. "Low Power Gated Bus Synthesis using Shortest-Path Steiner Graph for System-on-Chip Communications" *ACM/IEEE Design Automation Conference*, 2009.

Renshen Wang and Chung-Kuan Cheng. "On the Complexity of Graph Cuboidal Dual Problems for 3-D Floorplanning of Integrated Circuit Design" *ACM Great Lake Symposium on Very-large-scale Integration*, 2009.

Renshen Wang and Chung-Kuan Cheng. "Octilinear Redistributive Layer Routing in Bump Arrays" *ACM Great Lake Symposium on Very-large-scale Integration*, 2009.

Renshen Wang, Takumi Okamoto and Chung-Kuan Cheng. "Symmetrical Buffer Placement in Clock Trees for Minimal Skew Immune to Global On-chip Variations" *IEEE International Conference on Computer Design*, 2009.

Renshen Wang, Evangeline F. Y. Young and Chung-Kuan Cheng. "Representing Topological Structures for 3-D Floorplanning" *IEEE International Conference on Communications, Circuits and Systems*, 2009.

Renshen Wang, Rui Shi and Chung-Kuan Cheng. "Layer Minimization of Escape Routing in Area Array Packaging" *IEEE/ACM International Conference on Computer-Aided Design*, November 5-9, 2006.

ABSTRACT OF THE DISSERTATION

Physical Planning to Embrace Interconnect Dominance in Power and Performance

by

Renshen Wang

Doctor of Philosophy in Computer Science

University of California, San Diego, 2010

Professor Chung-Kuan Cheng, Chair

The growth of computer performance by Moore's law is currently limited by power consumption and waste heat removal. As feature size scales down, dynamic power of integrated circuits (IC) is increasingly dominated by interconnect wires. Reducing this power has become a focusing target for computer architects and circuit designers. In this dissertation, we study on two phases in IC design flow, floorplanning and interconnecting, to address the challenges on interconnect latency and power.

The first and most direct way is to shorten the interconnect lengths. Current IC chips are manufactured on a 2-dimensional silicon die. Moving to 3-dimensional enables us to make the same circuit within a much smaller foot print, and therefore greatly reduce interconnect power. For 3-D floorplanning, we study the computational complexity of placing cuboids in 3-dimensional space under adjacency constraints. We use the basic

graph-to-floorplan formulation and a 2.5-D variation with more strict layer constraints. In both cases, we find the problem is NP-complete by reductions from known NP-hard problems. The results, combined with previous work on the 2-D counterpart, show the fundamental hardness residing in 3-D structures, and reveal possible challenges in design complexity moving from 2-D to 3-D.

Another way, when interconnect distances can no longer be reduced, is to minimize the wires involved in communication actions. Most multi-component system-on-chips use bus or bus matrix to connect all the sub-systems. Complying with standard bus protocols, we devise a bus matrix synthesis scheme to minimize dynamic power on communications, while maintaining the bandwidth capability and routing resources. The geometric basis is the shortest-path Steiner graph extended from Steiner arborescence (shortest-path Steiner tree), where optimized graph structures are used to replace the large wire nets in traditional bus architectures. Experimental results show that near-optimal communication power can be achieved without consuming excessive on-chip resources.

Chapter 1

Introduction

Computer performance has been steadily growing as Moore's law [Moo65] provides larger silicon resources every generation, while processors gain more transistors and higher complexity. However, till recent years, computer chips hit a "power wall", i.e., the power consumption and waste heat will become unacceptable if processor design scales up in the same way as before. As a result, processor cores stop scaling up, instead increase in numbers to take advantage of the improving process technology. Since then, low power design has become a high priority objective in every aspect of the processor design ecosystems: from computer architecture to circuit design and manufacturing.

Physical design of integrated circuits determines the the location of each device on the chip, as well as the routing and buffering of each wire net. This design phase is performed after the function design, architecture design and logic design, usually the last phase before verification, testing and fabrication. The importance of this phase is significant and is still increasing, because circuit performance and power consumption are nowadays dominated by interconnect wires, whose geometry and electric properties are decided in this phase. For instance, clock rate is dictated by the signal propagation delay on long wires, dynamic power is mostly induced by wire capacitance, and wire length is determined by component placement.

In this dissertation, we study two stages in the physical design phase, floorplanning and bus interconnecting. In both stages, we look at the circuit from a relatively high level, i.e., we see mostly function blocks instead of logic gates. Physical design tasks are formulated into combinatorial problems on rectangular/cuboidal blocks and line segments. Algorithms and complexity on the problems are discussed and analyzed.

1.1 Floorplanning in Physical Design

1.1.1 2-D floorplanning

Floorplanning is an early stage in the hierarchical approach to chip design. Traditionally, integrated circuits are made on a single layer of silicon, and a floorplan can be a partition of the whole chip area into axis aligned rectangles, which are to be occupied by major functional blocks of the circuit. This partition is subject to various constraints and requirements of optimization: block area, aspect ratios, estimated total measure of interconnects, etc.

Most constraints or objectives make it NP-Hard to find the optimal floorplan. For instance, minimum total area can be reduced to the knapsack problem [Kar72] even when restricted into multiple 1-dimensional slices. In practical applications, metaheuristics like simulated annealing are usually applied on floorplan representations to search for relatively good, and often sub-optimal solutions.

We study a basic formulation of floorplanning proposed in [KK84] where it is called "rectangular dualization". A *rectangular dual* of graph G is defined as a rectangular dissection D which has its adjacency graph isomorphic to G. Figure 1.1 shows an example of a graph and its rectangular dual. Apparently, the adjacency graph of a rectangular dissection must be planar.

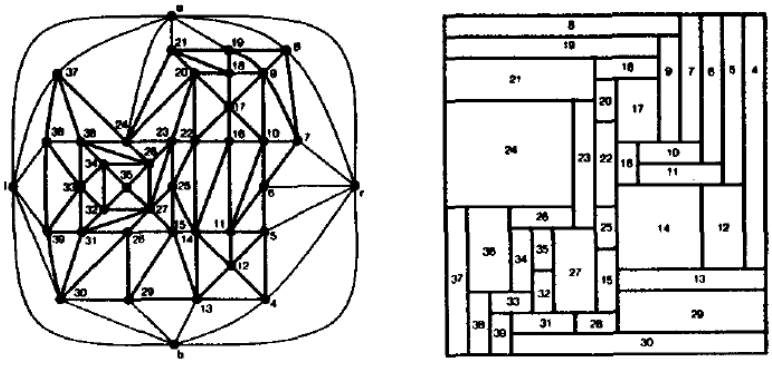

Figure 1.1: 2-dimensional rectangular dual

It is proved in [KK84] that a triangulated plane graph G has rectangular dual if and only if there exists a 4-completion of G. The 4-completion of plane graph G is defined as a plane graph H such that:

(a) the exterior face of H has 4 vertices;

(b) each interior face of H has 3 vertices;

(c) each non-face cycle in H has length ≥ 4;

(d) G can be obtained by deleting the 4 outer vertices on the exterior face.

Also a linear time algorithm is introduced in [BS86] which can find a rectangular dual of a triangulated graph. Although the practical influence of this formulation is limited, it is easy to solve and may provide an initial floorplan of the circuit as a starting point of placement optimizations. For instance, if a pair of blocks b_i, b_j are heavily connected, we let $(v_i, v_j) \in E$; or if both have high heat density, we let $(v_i, v_j) \notin E$.

1.1.2 Moving to 3-D

As technology advances, interconnect wires are dominating dynamic power consumption of integrated circuits. Moreover, [HMH01] shows that long wires (global interconnections) between system components are relatively scaling up. To resolve this bottleneck, new technologies such as 3-dimensional integration is a possible solution. Ideally, by using n-layer circuit we reduce the interconnect length by a factor of $\sqrt{n}$. However, the added dimension may also bring higher complexity and difficulty in design, CAD tools and fabrications. To fully exploit the advantages of 3-D circuits, we need to measure and understand the complexity it brings.

In literatures, 3-D floorplans include two classes. One is full 3-D, where function blocks are cuboids being placed in space with no distinguishable "layers" [YSNK00]; another one is multi-layer floorplan (also called 2.5-D), which is virtually a stack of 2-D floorplans with identical thickness [YHH06]. Current developing 3-D circuits are mostly 2.5-D, but we believe full 3-D floorplans will be adopted in future.

By extending the 2-D rectangular dual problem, we discuss the two classes of 3-D floorplans by a "cuboidal dualization" problem. The basic formulation is: *Given a graph $G = (V, E)$, can we find a set of cuboids as V with contact relations as E?*

The 2-D version of this problem (i.e. all the cuboids must be placed on the floor) is similar to the rectangular dual in [KK84], except that we allow empty area not covered by rectangles. This difference is for convenience in the 3-D version, and does not change the difficulty of the 2-D version as proved in chapter 2.3.

By our studies, the cuboidal dual problem differentiates 2-D and 3-D space. The existence of rectangular dual can be decided by a set of conditions in [KK84], which can be extended to decide 2-D cuboidal dual. However in the 3-D version, we find that deciding the existence of cuboidal dual is NP-complete, which implies that the conditions

like in [KK84] do not exist. There is a possible analogy between cuboidal dual and graph coloring, where 2-colorability is easy to decide but 3-colorability is NP-complete [Kar72]. One extra dimension in space brings a fundamentally higher level of complexity. In fact we prove the NP-completeness of 3-D cuboidal dual just by the reduction from 3-colorability. For the 2.5-D version, although the problem looks very similar to 2-D, we find it NP-complete when the number of layers reaches 3. The hardness of all the versions are listed in the following table.

Table 1.1: k-dimensional cuboidal dual problems

Floorplan dimensions	Number of layers	Hardness of problem
2-D	1	P
2.5-D	2	*open*
2.5-D	≥ 3	NP-complete
Full 3-D	undefined	NP-complete

The results imply that the complexity of circuit floorplanning can be greatly increased when we extend the circuit on the third dimension, even if the extension is limited to just a few layers. Besides heat removal and fabrication, this can possibly be another challenge we will face to move from 2-D to 3-D.

1.2 System Integration using Bus Matrix

1.2.1 Bus and bus matrix

After the floorplanning stage, the function blocks have been placed over the chip area, and need to be connected together by on-chip communication solutions. Bus architectures are usually chosen as the solution to integrate all the function blocks, so that designers can pick existing function modules which are compatible to standard bus interfaces. Current industrial on-chip bus standards include ARM AMBA, IBM CoreConnect, Altera Avalon, etc.

Figure 1.2 shows the sketch of an AMBA AHB bus. The components (function blocks) are classified into master devices and slave devices. Masters are typically microprocessors, which can start a transaction with one slave at a time. Slave selected by the master responds passively. When conflicting requests come from multiple masters at the same time, arbiter will decide the order of services. The bus here with a single arbiter supports a double-way transaction each time. When multiple transactions are

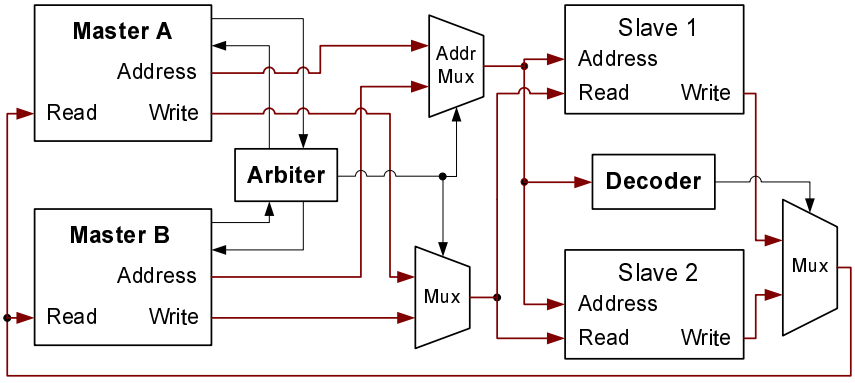

Figure 1.2: Sketch of an AMBA AHB bus

required for parallel processing, designers may choose a bus matrix, which can function as multiple copies of the basic bus.

Traditionally, on-chip or on-board buses consist of a small number of large wire nets, plus a few control units, because simplicity is usually preferred over other objectives. However, with the increasing power consumed by wires, especially by long wires which relatively scale up, power efficient on-chip communication architecture will become critical for overall system power. Simulation results in [LR04] suggest that a simple bus may consume 15% of total system power. Figure 1.2 also reveals that bus design has stayed on system-level, without consideration of the components' on-chip locations.

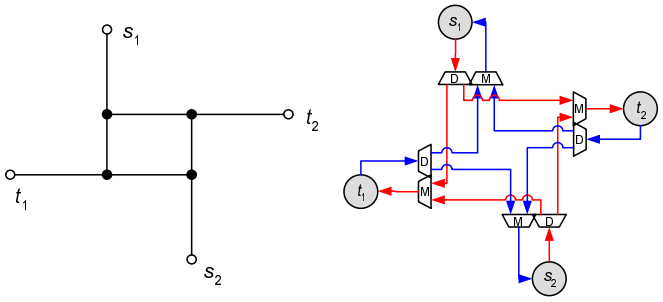

Figure 1.3: A shortest-path Steiner graph and its bus implementation

Since the power and performance of integrated circuits are dominated by interconnect wires, on-chip locations of system components are increasingly relevant to the ways of connecting them together. Therefore, we believe physical optimization has become a necessity to thoroughly exploit the design space to achieve best performance and power efficiency. In our studies, we devise a physical synthesis scheme of bus matrix based on shortest-path Steiner graph (example shown in figure 1.3), which can pro-

vide minimal power, adequate bandwidth capability, and use small amount of routing resources. Heuristics are devised to construct the shortest-path Steiner graph, and to reduce its scale with a minimal increment on path lengths. Experimental results on random test cases show the effectiveness of our scheme and algorithms. Similarities between our bus matrix results and city traffic planning further justify our methodology.

1.2.2 Bus matrix v.s. network-on-chip (NoC)

Bus architectures are currently the universal industrial solution for on-chip communications. Another type of network-on-chip (NoC) [DT01] communication has been studied by academia for nearly a decade, and is considered to be a good replacement of bus. Working like a miniaturized internet, a network-on-chip has no centralized control units, but consists of routers connected by wire links. We list a comparison among traditional simple bus, our optimized bus matrix and NoC in the following table.

Table 1.2: Comparisons among bus, bus matrix, and network-on-chip

	Unoptimized bus	Optimized bus matrix	Network-on-chip
Power	×	√	−
Latency	−	√	×
Bandwidth	×	−	√

On power consumption, a simple bus has very low power efficiency because of the large wire nets. By using a "bus gating" technique on the shortest-path Steiner graph, we let the transactions always take shortest distance and achieve the best efficiency. On NoC, while the transactions can be routed on short paths, the power overhead on routers and data packets are higher than that of bus matrix.

On latency, bus or bus matrix are always better than NoC because of the centralized control. The independency of routers in NoC leads to accumulated delay on routing computations, which inevitably leads to larger overall latency.

On bandwidth, NoC is generally better. Even when bus matrix has a comparable maximum bandwidth capability, NoC has better flexibility on bandwidth utilization because of the on-line routing mechanism.

By the comparison in three aspects, we believe bus architectures will continue to be widely applied in future on-chip systems, especially for those with latency-aware performance or strict power budgets.

Chapter 2

Complexity of Floorplanning in 3-D

2.1 Preliminaries and Formulations

Traditional 2-D floorplanning is to place a set of rectangles in a designated area to meet certain requirements. The basic constraint is that no common area can be shared by two or more rectangles. For 3-D, the problem becomes placing a set of cuboids in a space without common space shared by cuboids. A 2-D case can be regarded as a 3-D case with all the cuboids placed on the floor.

An adjacency graph can be constructed from a floorplan by assigning a vertex to each cuboid and add edge (v_i, v_j) when the two corresponding cuboids are contacting on surfaces. While this construction is easy, the reverse construction from graph to floorplan is not trivial. In [KK84] there is a set of sufficient and necessary conditions for a graph to be an adjacency graph of a rectangular dissection. The dissection is called a *rectangular dual* of the graph. For 3-D, we define a problem based on *cuboidal dual*s.

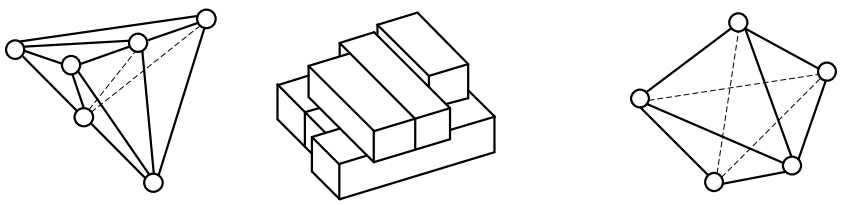

Figure 2.1: Graph-floorplan relations

A general 3-D cuboidal dual of an n-vertex graph $G = (V, E)$ is defined as a set of axis-parallel cuboids, each cuboid C_i corresponds to a vertex $v_i \in V$. No two cuboids share a common part of space. C_i and C_j are adjacent (contacting on a surface by a non-zero area) if and only if $(v_i, v_j) \in E$. Figure 2.1 shows a 6-vertex graph and one of its cuboidal duals, and a 5-vertex clique has no cuboidal duals.

A 2.5-D cuboidal dual is defined as a 3-D cuboidal dual with an additional constraint that every cuboid has height interval $[l-1, l]$, where l is an integer indicating the layer this cuboid is in.

A 2-D cuboidal dual is defined as a 2.5-D cuboidal dual with one layer, i.e., every cuboid is placed in height interval [0,1]. It is different from a rectangular dual [KK84] in that the set of cuboids can be a *subset* of a space dissection, so empty space is allowed.

Our basic problem is to find a cuboidal dual of a given graph G. For any of the 2-D, 2.5-D or 3-D case, the problem is trivially in NP, because it takes polynomial time to verify whether a given set of cuboids is a solution.

2.2 3-D Cuboidal Dual of General Graphs

To decide whether a graph has a 3-D cuboidal dual is NP-hard. We prove this by reducing the 3-colorability problem to 3-D cuboidal dual. It is well known that 3-colorability is NP-complete [Kar72]. We construct G from a 3-colorability instance $G_{3C} = (W, E')$. In the first step, we introduce a gadget of 7 vertices for each vertex in W, shown in figure 2.2.

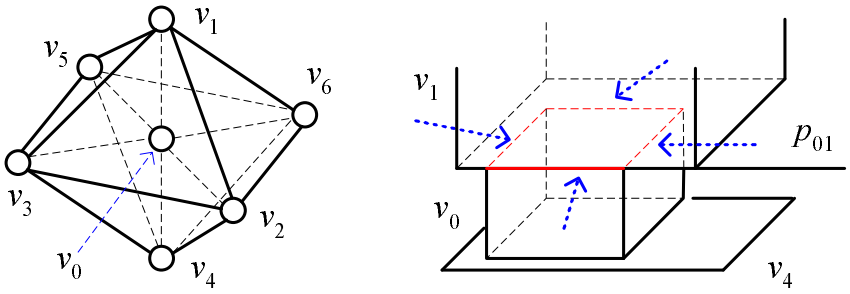

Figure 2.2: 7-vertex gadget and part of its cuboidal dual

The 7 vertices together with the edges form an octahedron composed of 8 small tetrahedrons. There is a cuboidal dual of this graph, and the contact surfaces between different pairs of cuboids are not independent of each other.

Lemma 1 *In the cuboidal dual of the 7-vertex gadget, the cuboids of two opposite vertices on the octahedron (e.g. v_1, v_4) are on opposite sides of the central cuboid (v_0).*

PROOF. Since cuboid v_0 and cuboid v_1 are adjacent, they are contacting on a common plane (we denote this plane as p_{01}). Their contacting area on p_{01} must be a rectangle, denoted as R (red in figure 2.2). Since rectangle R is the only common part of the two cuboids, by the convexity of cuboid, another cuboid (such as v_2) contacting both v_0 and v_1 must be on the boundary of R. Cuboid v_2, v_3, v_5, v_6 are four such cuboids in the gadget, with a closed loop $v_2 - v_3 - v_5 - v_6 - v_2$ of edges. The only way to place v_2, v_3, v_5, v_6 is surrounding the boundary of R, because R's inside area is occupied by v_0 and v_1.

Now we project all the seven cuboids onto p_{01}. We have v_2, v_3, v_5, v_6 surrounding rectangle R. Since v_4 is adjacent to these four vertices, the projection of cuboid v_4 on p_{01} must overlap with the area surrounding R, i.e., the projection of v_4 covers rectangle R. So on cuboid v_0's six surfaces, cuboid v_4:

• cannot be on the same surface as v_1 because v_4's projection on p_{01} overlap with R (which is part of v_1's projection);

• cannot be on the four adjacent surfaces of v_1's side because v_4's projection on p_{01} overlap with R (which is part of v_0's projection);

In conclusion, cuboid v_4 can only be on the surface of v_1's opposite side. □

Lemma 1 implies that the contacting directions of $v_1 \to v_0$ and $v_0 \to v_4$ are same, and we denote this direction as $d_{1,4}$.

Definition 1 *For a 7-vertex gadget N, $d_{1,4}(N)$ denote the axis (among x, y, z) whose orthogonal plane has overlapping projections from cuboids v_1 and v_4.*

In the same way of lemma 1, the other two pairs of vertices (v_2, v_5) and (v_3, v_6) also have cuboids on opposite sides of v_0. Another conclusion from figure 2.2 is that the cuboids of $v_1, v_2, \cdots, v_6$ cover all the 6 surfaces of cuboid v_0, proved as follows.

Lemma 2 *In the cuboidal dual of the 7-vertex gadget, the six cuboids of $v_1, \cdots, v_6$ each entirely covers one surface of cuboid v_0.*

PROOF. Cuboid v_2 is contacting cuboid v_4, which by lemma 1 is on the opposite side of cuboid v_1's surface. By the convexity of cuboid, v_2 cannot be on the same surface

of v_1. The same holds for v_3, v_5 and v_6. Thus, cuboid v_1 does not share v_0's surface with any other cuboid. By the symmetry of the gadget, the six cuboids do not share any of v_0's six surfaces, so each covers one. And since each cuboid is contacting four neighbors on adjacent surfaces, it must reach the four edges of its side of v_0, therefore entirely covers the surface. □

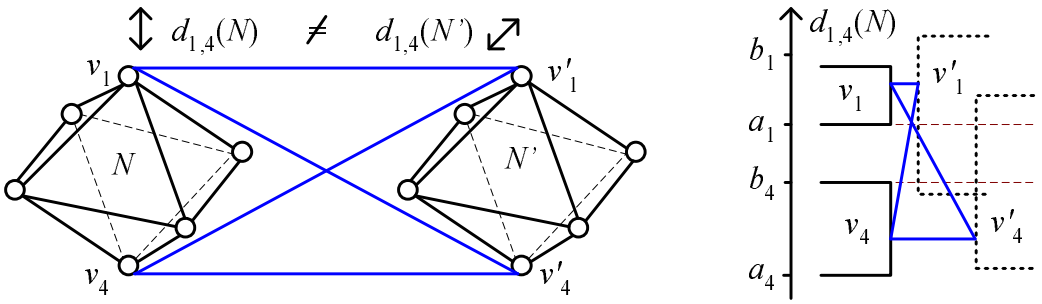

Figure 2.3: Enforcing 2 gadgets in different directions

Regarding the coordinate axis that $d_{1,4}(N)$ is parallel to, it has three possible directions: x, y and z. These directions can be used as the 3 possible colors in the 3-colorability problem, where a gadget N, representing a vertex w in $G_{3C} = (W, E')$, is colored as $d_{1,4}(N)$. The constraint of 3-colorability is that for edge $(w, w') \in E'$ in G_{3C}, the two vertices cannot share the same color. This constraint can be implemented as a bi-clique connection between v_1 and v_4 of two gadgets N and N'. As figure 2.3 shows, we look at an axis parallel to direction $d_{1,4}(N)$, where v_1 occupies interval $[a_1, b_1]$ and v_4 occupies interval $[a_4, b_4]$. If there is a bi-clique connection between $\{v_1, v_4\}$ and $\{v'_1, v'_4\}$, then both v'_1 and v'_4 must cover the interval between v_1 and v_4, which is $[b_4, a_1]$ on the axis, because the cuboids of v'_1 and v'_4 are contacting v_1 and v_4. So the two cuboids of v'_1 and v'_4 have an overlapping interval on this axis, which means $d_{1,4}(N')$ must be on a different direction, and thus $d_{1,4}(N') \neq d_{1,4}(N)$.

To complete the reduction from 3-colorability, we need to construct G based on the gadget nodes. We add 6 more vertices to the 7-vertex gadget to further restrict the directions of contacting surfaces among the cuboids of $v_1, \cdots, v_6$.

Lemma 3 *(Figure 2.4) Adding 3 pairs of vertices to the 7-vertex gadget:*

pair 1 connected to $\{v_1, v_2, v_4\}$ *and* $\{v_1, v_5, v_4\}$, *(green)*
pair 2 connected to $\{v_2, v_3, v_5\}$ *and* $\{v_2, v_6, v_5\}$, *(blue)*
pair 3 connected to $\{v_3, v_1, v_6\}$ *and* $\{v_3, v_4, v_6\}$, *(red)*

then *cuboids* v_1 *and* v_4 *have same width as cuboid* v_0 *(along* $d_{3,6}$*),*
cuboids v_2 *and* v_5 *have same height as cuboid* v_0 *(along* $d_{1,4}$*),*
cuboids v_3 *and* v_6 *have same length as cuboid* v_0 *(along* $d_{2,5}$*).*

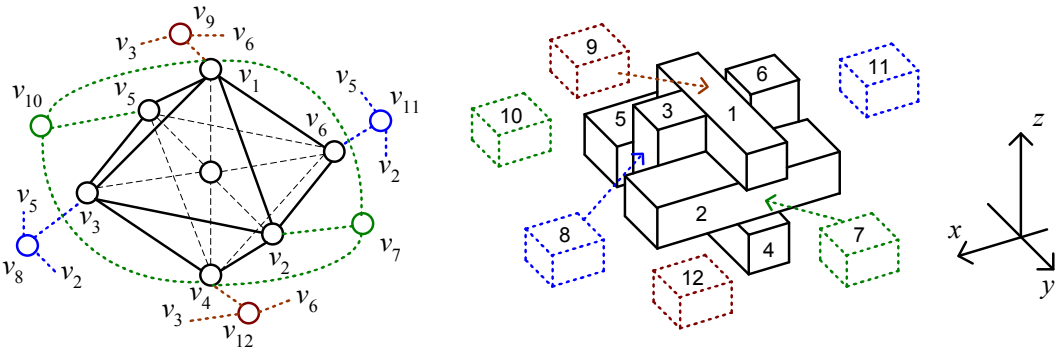

Figure 2.4: 13-vertex gadget and its cuboidal dual

PROOF. As figure 2.4 shows, we already know from lemma 1 and lemma 2 that the six cuboids of $v_1, \cdots, v_6$ are each placed with pairs (v_1, v_4), (v_2, v_5), (v_3, v_6) on opposite sides of cuboid v_0, and the six surfaces of cuboid v_0 are covered by $v_1, \cdots, v_6$. In this coordinate system, $d_{1,4}$ of the gadget is parallel to axis z.

Consider the cuboid of v_7 contacting three cuboids of v_1, v_2 and v_4. It must contact cuboid v_2 on the $+y$ side, because:

• cuboid v_2 and v_7 are contacting both v_1 and v_4, so on their projections to axis z (or $d_{1,4}$), v_7 overlaps with v_2, which means v_7 cannot be at $+z$ or $-z$ side of v_2;

• if cuboid v_7 is contacting cuboid v_2 on the $-y$ side, without loosing generality we assume it is on $+x$ side of cuboid v_3, v_0, and v_6. To make cuboid v_7 contacting cuboid v_1 and v_4, we need $x_{1,+} \geq x_{3,+}$ and $x_{4,+} \geq x_{3,+}$ (denoting $x_{k,+}$ as the x coordinate of cuboid k's $+x$ surface). Plus cuboid v_8 is contacting cuboid v_2, v_3, and v_5, we have $x_{5,+} \geq x_{3,+}$. So in the projection on xy plane, rectangle v_3 is completely contained in rectangle v_1 and rectangle v_4. Thus, cuboid v_3 has only the $+x$ surface not covered. But it is impossible to make cuboid v_9 on v_3's $+x$ surface while also contacting cuboid v_1 and v_6, which leads to contradiction.

• if cuboid v_7 is contacting cuboid v_2 on $+x$ direction (the case on $-x$ direction is symmetric), then cuboid v_2 cannot protrude among cuboid v_1 and v_4 on $+x$ direction, i.e., we have $x_{2,+} \leq x_{1,+}$ and $x_{2,+} \leq x_{4,+}$. So when cuboid v_8 is contacting cuboid v_2, v_3, and v_5, it cannot be contacting the $+z$ or $-z$ surface of v_3, so must be contacting

v_3's $+x$ surface, which leads to $x_{2,+} \geq x_{8,-} = x_{3,+}$, and therefore we have $x_{1,+} \geq x_{3,+}$ and $x_{4,+} \geq x_{3,+}$. Hence, rectangle v_3 is also completely contained in rectangle v_1 and rectangle v_4 in the projection on xy plane, which leads to the same contradiction as above.

By symmetry, cuboid v_9 must be on $-y$ side of cuboid v_5, cuboid v_8 must be on $+x$ side of cuboid v_3, cuboid v_{11} must be on $-x$ side of cuboid v_6, cuboid v_9 must be on $+z$ side of cuboid v_1, and cuboid v_{12} must be on $-z$ side of cuboid v_4. Thus, the contacting surfaces among the cuboids of $v_1, \cdots, v_6$ must be in exactly the same topology as in figure 2.4. $\square$

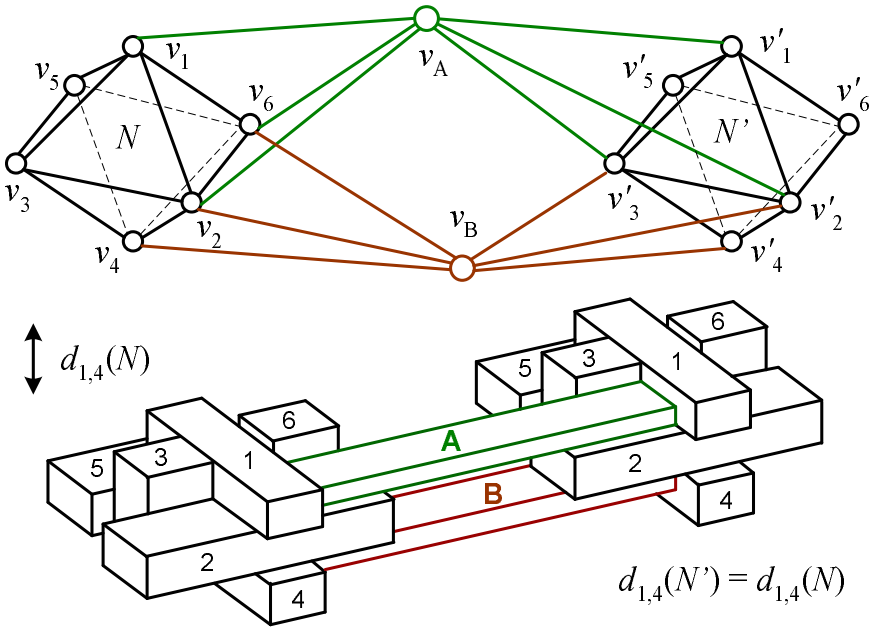

Figure 2.5: Two 13-vertex gadgets N and N' with $d_{1,4}(N)$ and $d_{1,4}(N')$ aligned to the same direction (2-alignment)

By constructing this 13-vertex gadget, the original 7-vertex gadget has a definite shape, and we can easily align multiple gadgets into the same direction with some additional vertices in G as follows.

In figure 2.5 we use a simplified octahedron to represent each 13-vertex gadget, and add two vertices v_A and v_B. Consider their connections with gadget N' on right. Since v_A is simultaneously contacting v'_1, v'_2 and v'_3, by lemma 3 and figure 2.4, cuboid v_A must be on the corner formed by the 3 cuboids, and is therefore contacting v'_2 from above. Similarly, v_B is contacting v'_4, v'_2 and v'_3, so cuboid v_B must be on the corner and contacting v'_2 from below. As a result, the direction $v_A \to v_B$ is same as $d_{1,4}(N')$.

The same conclusion can be found on gadget N, i.e., the direction $v_A \rightarrow v_B$ is same as $d_{1,4}(N)$. Thus, with two additional vertices we make $d_{1,4}(N) = d_{1,4}(N')$.

Besides the alignment of $d_{1,4}$, we also need an alignment such that the 3 directions $d_{1,4}$, $d_{2,5}$ and $d_{3,6}$ of the two gadgets are all in parallel.

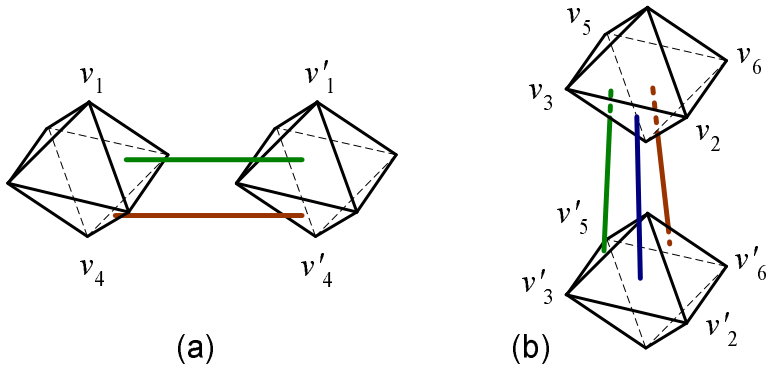

Figure 2.6: 2-alignment and 3(complete)-alignment

Figure 2.6(a) is the simplified notation of the alignment illustrated in figure 2.5, where only the directions of $d_{1,4}(N)$ and $d_{1,4}(N')$ are parallelized. We call this a 2-alignment. In figure 2.6(b) there are 3 additional vertices (called 3-alignment), the result is $d_{2,5}(N) = d_{2,5}(N')$ and $d_{3,6}(N) = d_{3,6}(N')$, which also implies $d_{1,4}(N) = d_{1,4}(N')$. So in a 3-alignment, the two gadgets are aligned in every direction.

Also notice that the direction of displacement from one gadget to the other in a 2-alignment is along $d_{2,5}(N)$ or $d_{3,6}(N)$, while the displacement in a 3-alignment must be along $d_{1,4}(N)$. These two cases of alignment enable the alignment of a pair of 13-vertex gadgets N and N' along any of the $\{x, y, z\}$ axes, and always guarantee $d_{1,4}(N) = d_{1,4}(N')$. Based on these cases, we can construct large sets of 13-vertex gadgets all aligned in the same direction, each set acting as a single vertex in $G_{3C} = (W, E')$, which help to complete the reduction from 3-colorability.

Theorem 1 *3-colorability reduces to 3-D cuboidal dual.*

Proof. We use the 13-vertex gadget as a vertex cluster node in our construction of G. Given a graph of 3-colorability $G_{3C} = (W, E')$ with n vertices denoted as $w_1, w_2, \cdots, w_n$, for each vertex w_i, we construct n 13-vertex gadget nodes in G, denoted as $s_{i,1}, \cdots, s_{i,n}$, sequentially connected by 2-alignments. Then for each gadget node $s_{i,j}$, construct 4 auxiliary gadgets as follows: $s_{i,j}$ 2-aligns with $t_{1,i,j}$, $t_{1,i,j}$ 3-aligns with $t_{2,i,j}$, $t_{2,i,j}$ 2-aligns with $t_{3,i,j}$, and finally $t_{3,i,j}$ 2-aligns with $u_{i,j}$.

For each edge $(w_i, w_j) \in E'$, we pick gadget nodes $u_{i,j}$ and $u_{j,i}$, connect v_1 and v_4 of $u_{i,j}$ with v_1 and v_4 of $u_{j,i}$, so that the two gadgets $u_{i,j}$ and $u_{j,i}$ are enforced in different directions (as in figure 2.3). In this way, graph G has a cuboidal dual *if and only if* G_{3C} is 3-colorable.

The "if and only if" holds because of the gadget sets in G constructed from the vertices of G_{3C}. Assume G_{3C} has n vertices. On each vertex w_i of G_{3C}, we put $5n$ gadgets $s_{i,1} \sim s_{i,n}$, $t_{1,i,1} \sim t_{1,i,n}$, $t_{2,i,1} \sim t_{2,i,n}$, $t_{3,i,1} \sim t_{3,i,n}$ and $u_{i,1} \sim u_{i,n}$. The $d_{1,4}$ direction of all these gadgets are the same, because they are connected by either 2-alignments or 3-alignments. Denote these $5n$ gadgets as set S_i, and the direction $d_{1,4}(S_i)$ has 3 choices x, y or z, just like the color of vertex w_i in G_{3C} has 3 choices. And by enforcing $u_{i,j}$ and $u_{j,i}$ in different directions for edge $(w_i, w_j) \in E'$, we also enforce set S_i and S_j in different directions.

From left to right, if graph G has a cuboidal dual, we color each vertex w_i by the $d_{1,4}$ direction of gadgets in S_i. By the alignments and enforcements among gadgets in G, w_i and w_j have different colors whenever there is an edge between w_i and w_j. Therefore, we have a valid 3-coloring on G_{3C}.

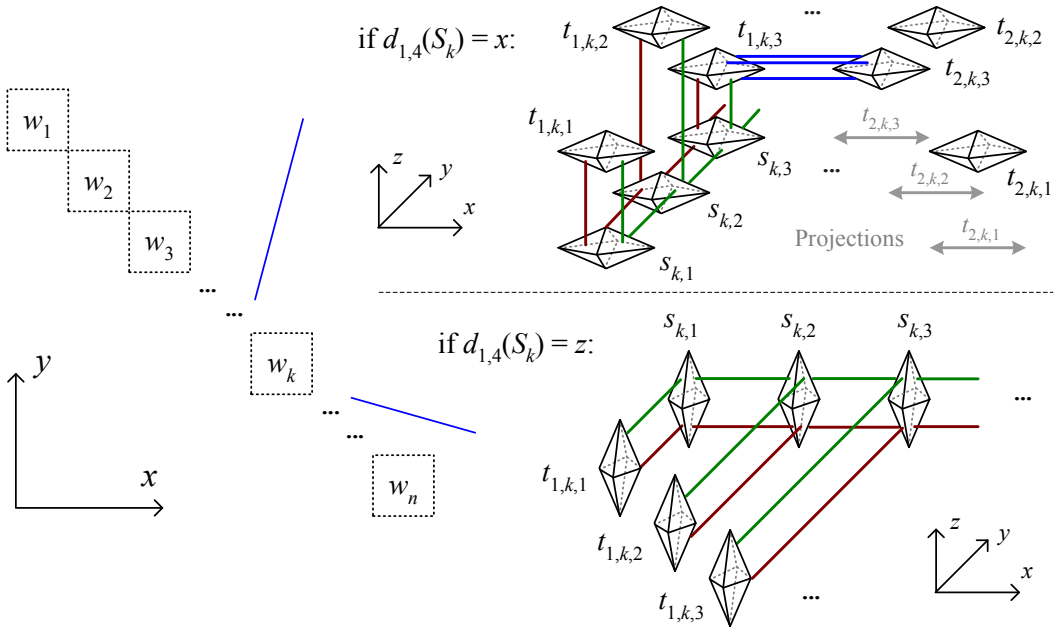

Figure 2.7: Construction of cuboidal dual from G when G_{3C} is 3-colorable

On the other hand, if G_{3C} is 3-colorable, then we can construct a cuboidal dual according to figure 2.7. The $s_{i,j}$ gadgets of G_{3C}'s vertices $w_1, \cdots, w_n$ are placed on the

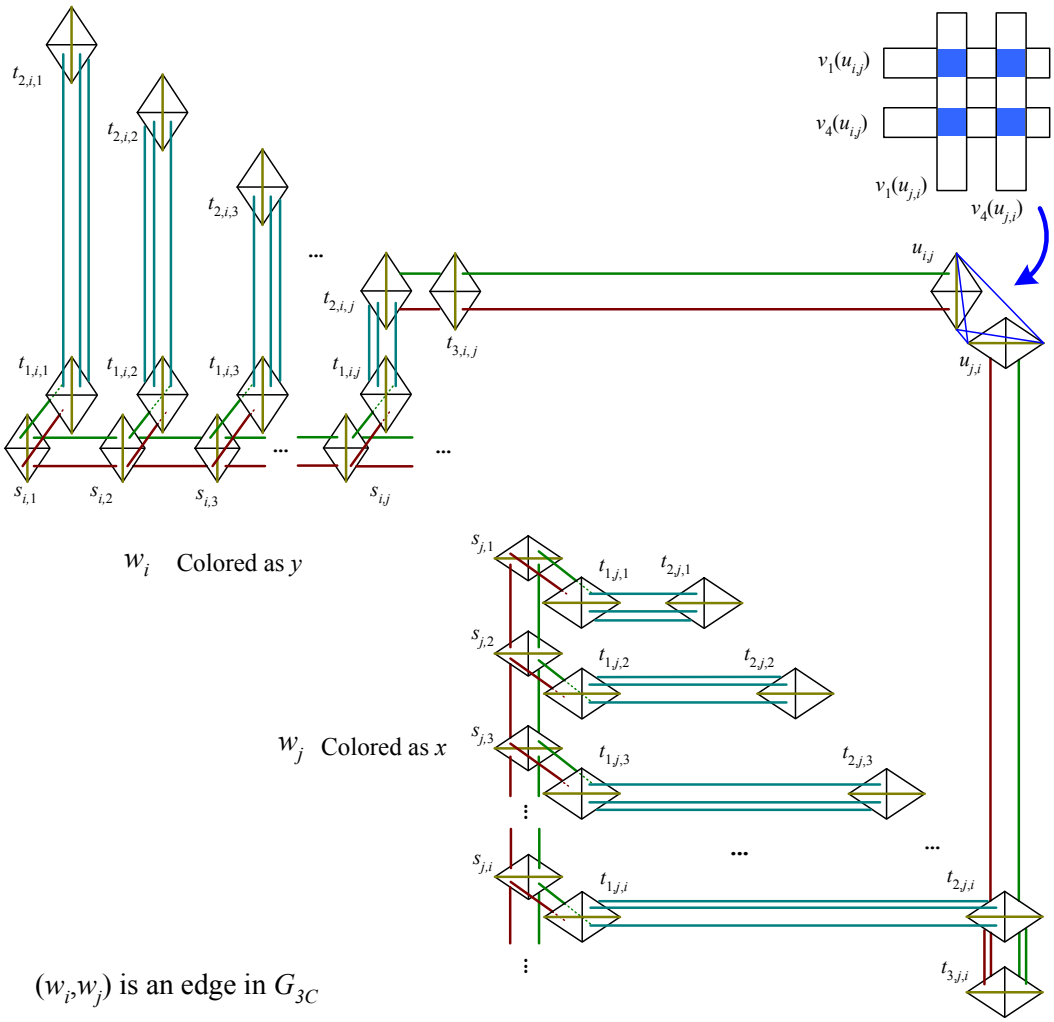

Figure 2.8: Connection in the cuboidal dual for an edge in G_{3C}

xy-plane and with a top view of figure 2.7's left part. Each G_{3C}'s vertex w_k has a square area, and is assigned with a direction in $\{x, y, z\}$ according to the coloring of G_{3C}, which decides the direction of gadget set $d_{1,4}(S_k)$. Every edge (w_i, w_j) in E' has a connection through gadgets as constructed in G. We assign a unique height $(H_{i,j})$ to every edge, so that out of the square areas of w_i and w_j, at height $H_{i,j}$ there are only connection cuboids between w_i and w_j. Thus, we guarantee different edges can have series of gadget connections constructed without conflict. Details of the gadgets' placement are revealed in figure 2.7 and 2.8, explained as follows.

(i) If $d_{1,4}(s_{i,1})$ is parallel to z, the auxiliary gadgets $\{t_{1,i,j}\}$ can be placed along a 45° line, and by 3-alignments each $t_{2,i,j}$ is leveraged to the height of $H_{i,j}$;

(ii) Otherwise $d_{1,4}(s_{i,1})$ is parallel to x or y, then each $t_{1,i,j}$ is leveraged to the height of $H_{i,j}$, and by 3-alignments the gadgets of $\{t_{2,i,j}\}$ are by top view placed along a 45° line.

Therefore, by the layout of figure 2.7, auxiliary gadgets $\{t_{2,i,j}\}$ can always be placed along a 45° line by top view. Then for any i, j such that there is an edge $(w_i, w_j) \in E'$ with $d_{1,4}(s_{i,1}) \neq d_{1,4}(s_{j,1})$, we can always construct $t_{2,i,j} \rightarrow t_{3,i,j} \rightarrow u_{i,j}$ along x, $t_{2,j,i} \rightarrow t_{3,j,i} \rightarrow u_{j,i}$ along y, or vice versa. These connection gadgets are on the height of $H_{i,j}$, which prevents conflicts with other edges. So as shown in figure 2.8, $u_{i,j}$ and $u_{j,i}$ can finally make a connection at the intersecting point and form the biclique of $\{v_1(u_{i,j}), v_4(u_{i,j})\}$ and $\{v_1(u_{j,i}), v_4(u_{j,i})\}$. In this way we construct the edge connections, and the cuboidal dual construction of G is complete.

The reduction is thus proved. This is a polynomial reduction, because for each vertex in G_{3C}, we construct $5n$ gadgets in G, therefore graph G's size is on $O(n^2)$ scale of G_{3C}'s size. □

Corollary 1 *The problem of finding a graph's 3-D cuboidal dual is NP-complete.*

2.3 2.5-D Cuboidal Dual of Layered Graphs

In the last section we show that general 3-D cuboidal dual is hard. Now we look at the 2.5-D version of the problem which looks less complex. We start with the basic single-layer cuboidal dual of a planar graph G.

2.3.1 Single layer (2-D) cuboidal dual

The 2-D "rectangular dual" problem is first studied in [KK84] and [BS86]. By using a 4-completion graph, a simple rule to decide if a graph G has a rectangular dual is Theorem 1 of [KK84]: *A planar graph G drawn on a plane with all triangular interior faces has a rectangular dual if and only if there exists a 4-completion of G.* A *4-completion graph* is a graph H drawn on a plane such that:

(a) the exterior face of H has 4 vertices;

(b) each interior face of H has 3 vertices;

(c) each cycle in H that is not a face has length≥ 4;

A 4-completion of G is a 4-completion graph H that G can be obtained by deleting the 4 exterior vertices of H.

On our definition of cuboidal duals which allows empty space between cuboids, the deciding rule becomes more general and simplified.

Theorem 2 *A graph G has a 2-D cuboidal dual if and only if G can be drawn planar with no 3-vertex cycle containing interior vertex (vertices).*

PROOF. Given a planar graph G drawn without non-empty 3-vertex cycles, we add 4 exterior vertices v_n, v_s, v_w, v_e, and then add edges so that the resulting graph H is a 4-completion graph.

Starting from the exterior 4-vertex cycle of v_n, v_s, v_w, v_e, we first make the graph 2-connected. For any vertex or 2-connected component v inside a cycle C_1 with at least 4 vertices, (figure 2.9(a))

(i) If v is 1-connected to C_1 by vertex v_1, we find another vertex v_2 on C_1 with no edge to v_1. Such v_2 exists, otherwise every vertex is connected to v_1, then either C_1 is triangular, or C_1 itself is contained in a 3-vertex cycle, and both situations contradict with our given condition. Add an edge (v, v_2), which makes v 2-connected to C_1. C_1 is then decomposed into 2 smaller cycles, both with length ≥ 4 since there is no edge (v_1, v_2);

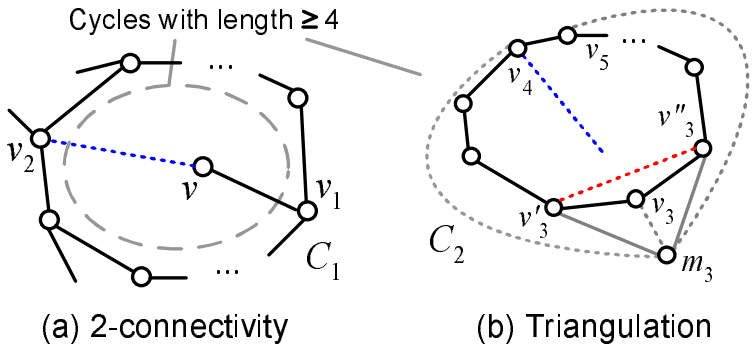

Figure 2.9: Triangulation without producing non-empty 3-vertex cycles

(ii) If v is not connected to C_1, pick any vertex v_1 on C_1 and add edge (v, v_1). v and C_1 then become 1-connected, and 2-connectivity can be achieved by case (i).

After the graph is 2-connected, every face is a cycle with no repeated vertices and no interior vertices. We then triangulate every face with more than 3 vertices, so that every face except the outer boundary is a triangle.

As shown in figure 2.9(b), for a face cycle with length ≥ 4, denoted it as C_2 and pick any vertex v_3 on C_2. The two neighbors of v_3 on C_2 are denoted as v_3' and v_3''.

(i) If there is vertex v_4 on C_2 not connected with v_3, and there is no vertex out of C_2 connected with both v_3 and v_4, then we add edge (v_3, v_4), which does not induce non-empty 3-vertex cycles;

(ii) Otherwise, add edge (v_3', v_3''). In this case every vertex on C_2 except $\{v_3, v_3', v_3''\}$ has a common neighbor with v_3. If v_3' and v_3'' have a common neighbor m_3, then all the common neighbors of v_3 and other vertices must also be m_3, and we have a 3-vertex cycle $m_3 \rightarrow v_4 \rightarrow v_5 \rightarrow m_3$ containing entire C_2, which contradicts the condition. So v_3' and v_3'' have no common neighbor out of C_2 and adding edge (v_3', v_3'') does not induce non-empty 3-vertex cycles.

Repeating this triangulation until the graph is completely triangulated, we get a 4-completion graph H. By Theorem 1 of [KK84] there is a rectangular dual of H. To obtain the 2-D cuboidal dual of the original graph G, we remove the 4 rectangles on the outer boundary which correspond to the 4 added vertices. For each edge (v_i, v_j) in H which is not in G, we add a gap between the contacting rectangles so that they are no longer contacting, and the final set of rectangles is the cuboidal dual of G.

The operation of adding gap can be done individually for each (v_i, v_j) in H but not in G. There are two cases, as shown in figure 2.10.

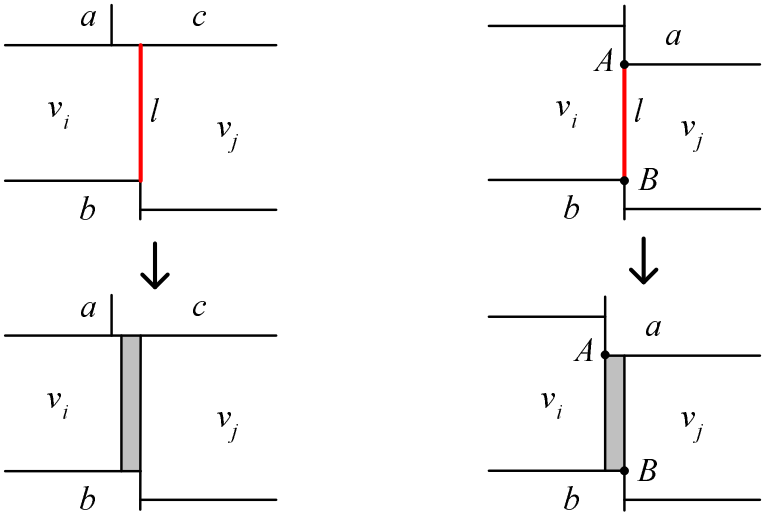

Figure 2.10: Adding gap between rectangles for edge not in G

(i) if v_i's contacting edge is contained in v_j's contacting edge, then we only move v_i's edge backward by ϵ. ϵ takes a value less than the minimum distance between any pair of existing points in the rectangle set, so that the only change in the contacting

topology is between v_i and v_j;

(ii) if neither contacting edge is contained in the other, we denote their common edge as l with end points AB (figure 2.10). For each edge on the infinite line of l, if it is above A or it is v_i's right edge, move it leftward by ϵ; if it is below B or it is v_j's left edge, no operation. Again ϵ is less than the minimum existing distance between points. The operation may involve rectangles other than v_i and v_j, but still the only change in topology is between v_i and v_j.

Finally, if graph G unavoidably contains a non-empty 3-vertex cycle, there is no 2-D cuboidal dual. Because as figure 2.11(a), the 3 contact surfaces of the 3 cuboids on the cycle form a "T" shape on the plane without internal space, so there is no room for the component(s) inside the 3-vertex cycle. □

When the given graph G is not drawn on a plane, the planar graph testing and embedding algorithm in [CNAO85] can be used to test planarity and embed G into the plane in linear time. If G is not planar, clearly there is no 2-D cuboidal dual. Otherwise G is planar, and we draw a plane and resolve (if possible) all the non-empty 3-vertex cycles one by one in G with following operations.

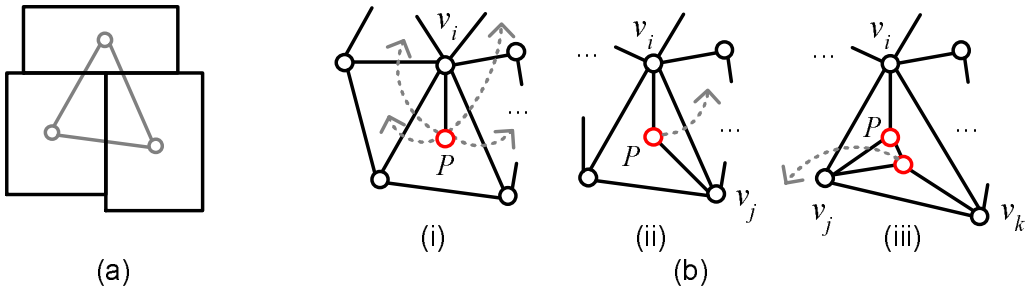

Figure 2.11: Empty 3-vertex cycle and operations to resolve non-empty 3-vertex cycle

Assuming G is connected (otherwise we run the algorithm on each connected component), for each non-empty 3-vertex cycle C with an interior component P, as illustrated in figure 2.11(b):

(i) if P is connected to one of the vertices on the cycle (denoted v_i), and there is another face on v_i with with cycle length ≥ 4, move P into that face;

(ii) if P is connected to two of the vertices on the cycle (denoted v_i, v_j), and there is another face F' containing both v_i and v_j with cycle length ≥ 4, there are two cases:

- if v_i's neighbor in P and v_j's neighbor in P are not the same vertex, move P

into face F';

• if v_i's neighbor in P and v_j's neighbor in P are the same vertex, only when face F'' on the other side of edge (v_i, v_j) has cycle length ≥ 4, move P into face F'';

(iii) if P is connected to all the three vertices on the cycle (denoted v_i, v_j and v_k), and there is another face F' with cycle length ≥ 4 and containing both edges of (v_i, v_j) and (v_j, v_k), also v_i's neighbor in P and v_k's neighbor in P are not the same vertex, then we can move P into face F'.

Repeating the three operations above, if all the non-empty 3-vertex cycles are resolved, a 2-D cuboidal dual exists. Otherwise, either in case (i), (ii) or (iii), there is either no adjacent face with cycle length ≥ 4 to place P, or $v_i \rightarrow P \rightarrow v_{j/k}$ form another 3-vertex cycle, there is no solution to draw G without a non-empty 3-vertex cycle, so G has no 2-D cuboidal dual.

Multiple algorithms of constructing a rectangular dual from G are introduced in [KK84] and [BS86], and [BS86] provides the constructions in linear time. Based on these algorithms, plus the transformation from G to a 4-completion graph, which can also be done in linear time, a 2-D cuboidal dual can be constructed from a planar graph G in linear time. Figure 2.12 shows an example of the construction.

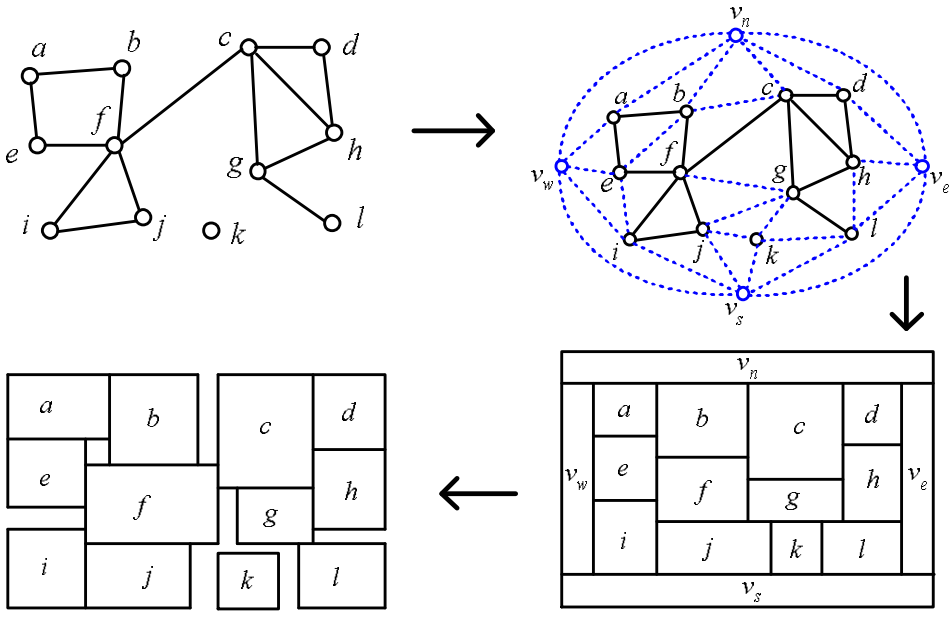

Figure 2.12: From a graph to its 2-D cuboidal dual

Corollary 2 *The problem of finding a graph's 2-D cuboidal dual is in P.*

2.3.2 3-layer 2.5-D cuboidal dual

With the 2-D cuboid dual problem (single layer case of 2.5-D) solved, we see how the difficulty of a problem differs from 2-D to 3-D, and the fundamental complexity on 3-D structures. To further explore the whole set of formulations, we look at something between these two versions – a multi-layer 2-D version. Multi-layer floorplans are usually called 2.5-D because of their characteristics between 2-D and 3-D. For this purpose, we first define a layered graph.

Definition 2 *In a k-layer graph $G = (V, E, L : V \mapsto \{1, \cdots, k\})$, each vertex is assigned with a layer between 1 and k, and each edge is either in a layer or between two consecutive layers, i.e., $(v_i, v_j) \in E \Rightarrow |L(v_i) - L(v_j)| \leq 1$.*

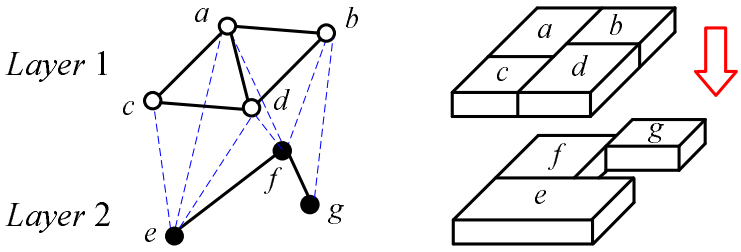

Figure 2.13: A 2-layer graph and 2.5-D cuboidal dual

In the problem here, we are given a layered graph $G = (V, E, L : V \to \{1, \cdots, k\})$. The 2.5-D cuboidal dual of G is a special 3-D cuboidal dual that the cuboid of v_i has fixed z interval of $[L(v_i) - 1, L(v_i)]$. Figure 2.13 shows an example of a 2-layer graph with its 2.5-D cuboidal dual.

The added constraints on cuboids and contacts lower the freedom of contacting directions. For edge (v_i, v_j), if v_i and v_j are on the same layer, their contacting direction has 2 choices, otherwise $|L(v_i) - L(v_j)| = 1$ and the contact surface must be parallel to the layer dividing planes. However, although the gadgets no longer have 3-dimensional freedom, we have other gadgets making the problem harder than the single layer case.

We find that when graph G has 3 layers, deciding its 2.5-D cuboidal dual is no less difficult than Planar 3-SAT, which is proved to be NP-complete in [Lic84]. 3-SAT is a basic NP-complete problem introduced in [Coo71]. A Planar 3-SAT instance has the same set of variables $U = \{u_1, \ldots, u_n\}$ and set of clauses $C = \{c_1, \ldots, c_m\}$ as 3-SAT. But in Planar 3-SAT, if we regard each variable or clause as a vertex and add edge (u_i, c_j) if clause c_j contains variable u_i, the resulting graph G_{p3SAT} is a planar graph.

Before we reduce Planar 3-SAT to 2.5-D cuboidal dual, we need a planar graph's rectilinear path embedding on a plane, which can be converted from its Fáry embedding (straight line embedding). For convenience, we use the straight line embedding on the $(n-2) \times (n-2)$ integer grid, which is from [Sch90].

Lemma 4 *Given a planar graph G_p with n vertices, we can place the set of vertices on a $(n-2) \times (n-2)$ integer grid, such that if each vertex v_i has a non-zero diameter, then each edge (v_i, v_j) has a rectilinear path from v_i to v_j with no more than $2n-4$ corners, and without intersections between paths.*

PROOF. By [Sch90], there is a straight line embedding of G_p on the $(n-2) \times (n-2)$ integer grid, so that all the vertices are on the grid and edges only intersect at end points.

Started with this embedding, for each edge which is neither horizontal nor vertical, we take its slope-intercept form $y = mx + b, x \in [x_l, x_r]$. By using the floor function ($\lfloor x \rfloor$ is the largest integer not greater than x), we turn this line into $y = \lfloor mx+b \rfloor$ which is a series of horizontal segments. Connecting these segments in order by vertical segments, the straight line is turned into a rectilinear path. Figure 2.14(a→b) is an example of this step.

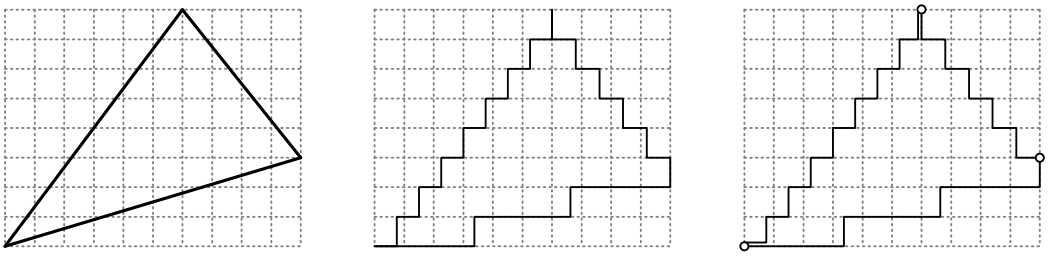

Figure 2.14: Converting a straight line embedding into a rectilinear path embedding

The rectilinear paths have following three properties:

(i) Each path has no more than $2n-4$ corners. Because the vertices are on a $(n-2) \times (n-2)$ integer grid, so any path from (x_1, y_1) to (x_2, y_2) goes through $|y_1 - y_2| \leq n-2$ vertical units. Each vertical unit segment has at most 2 corners, therefore the total number of corners does not exceed $2n-4$.

(ii) No crossing for any pair of paths (α, β) with $x_1, x_2 \in [x_\alpha^-, x_\alpha^+] \cap [x_\beta^-, x_\beta^+]$, $y_\alpha(x_1) < y_\beta(x_1)$, and $y_\alpha(x_2) > y_\beta(x_2)$. Because the floor function is monotonic, $\lfloor a \rfloor <$

$\lfloor b \rfloor$ necessarily leads to $a < b$, so a crossing between two paths leads to a crossing between the original straight lines, which is contradictive to the given planar graph embedding.

(iii) No identical paths, because each path has a unique pair of end points.

With edges converted into rectilinear paths, we then perform the following two operations.

(i) If a path intersects a vertex (because of y coordinates lowered by the floor function), then the original line must be above the vertex, so we move the horizontal intersecting segment upwards ($+y$) by ϵ. ϵ takes a small value (e.g. $1/n^2$ grid unit).

(ii) If two or more paths share segment(s), then we find the diverting point (it exists because no identical paths), and look at the shared segment on this point. If the segment is horizontal, then move the segments upward ($+y$) by ϵ on the set of paths which either turn upward or don't turn downward; if the segment is vertical, then move the segments leftward ($-x$) by ϵ on the set of paths which either turn leftward or don't turn rightward. Figure 2.14(b→c) is an example of this operation. Note if there are two diverting points, the operations based on them are consistent because there are no crossing paths.

Repeat these two operations, and the intersections can be all resolved. Because we only need limited number of spacings to avoid superposition of paths, and since the operations always move the segments towards $-x$ or $+y$, there is no dead loop in the repetition of operations.

In the end, we give each vertex a diameter to reach all its path segments that are moved upwards or leftwards, the result is a rectilinear path embedding of G_p, with each path no more than $2n - 4$ corners and no path intersections. □

Based on lemma 4, the reduction from Planar-3SAT to 2.5-D cuboidal dual can be constructed. We start with two basic gadgets shown in figure 2.15. In figure 2.15(a), if two vertices on layer i and two vertices on layer $i + 1$ are completely connected as a clique K_4, then in the cuboidal dual the contact surface between the two cuboids in layer i must be orthogonal to the one in layer $i + 1$. Because of the same reason as in figure 2.3, if the two pairs have the same direction, a complete connection is impossible. And in figure 2.15(b), the diamond gadget is similar to the 7-vertex gadget figure 2.2, so that the rectangles of v_1 and v_3 must be on opposite sides of the central rectangle.

For a clause c_i in Planar-3SAT, we can construct a 2-layer gadget as in figure 2.16(a), where the white vertices are on layer 1 and the black vertices are on layer

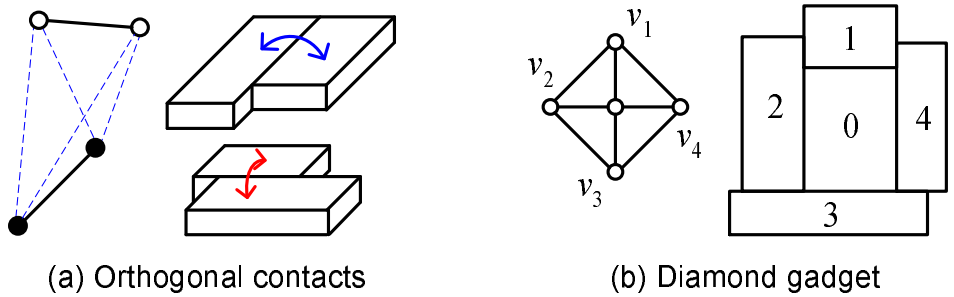

Figure 2.15: Basic gadgets in 2.5-D cuboidal dual

2. Two pairs of vertices p_1 and p_2 on layer 1 are enforced to have orthogonal contact surfaces by the diamond gadget on layer 2. Meanwhile the two pairs are also connected through three 6-vertex gadgets which have a different property as the diamond gadget: the contact directions of $v_0 \rightarrow v_1$, $v_0 \rightarrow v_2$, $v_0 \rightarrow v_3$ and $v_0 \rightarrow v_4$ have more freedom than in the diamond gadget, but are still dependent on each other. For instance, assume the direction of $v_0 \rightarrow v_4$ of such a gadget is determined like in figure 2.16(b), there are two cases.

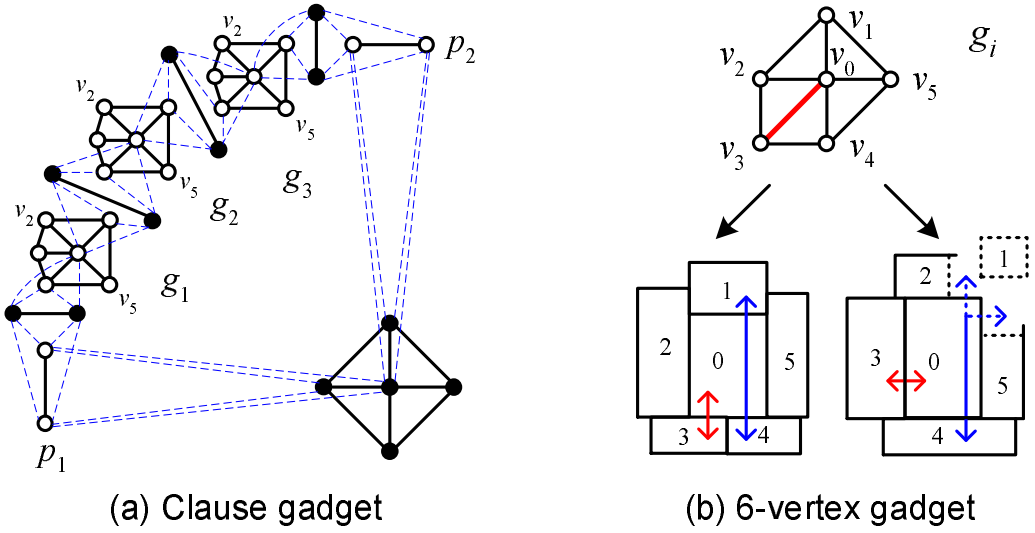

Figure 2.16: 2-layer subgraph of a clause gadget

(i) if $v_0 \rightarrow v_3$ is on the same direction, i.e., v_3 and v_4 are on the same side of central rectangle v_0, the 6-vertex gadget acts as a diamond gadget, so v_1 must be on the opposite side of v_0;

(ii) if v_3 is not on the same side of v_4, since this gadget has one more vertex than the diamond gadget, v_1 has the freedom of being on either of the two sides of v_0.

Lemma 5 *The clause gadget in figure 2.16(a) has a 2-layer cuboidal dual if and only if at least one 6-vertex gadget has $v_0 \to v_3$ horizontal.*

PROOF. Starting from the vertical pair p_1, which is connected to the horizontal pair p_2 through three 6-vertex gadgets. The first gadget g_1 has vertical $v_0 \to v_4$ due to the orthogonality enforcements from p_1. By the same enforcements, $v_0 \to v_1$ of g_1 is parallel to $v_0 \to v_4$ of g_2, and $v_0 \to v_1$ of g_2 is parallel to $v_0 \to v_4$ of g_3. Finally $v_0 \to v_1$ of g_3 is horizontal as enforced by vertex pair p_2.

If all the 6-vertex gadgets here have $v_0 \to v_3$ vertical: Starting at g_1, its $v_0 \to v_3$ and $v_0 \to v_4$ are both vertical, so v_3 and v_4 must be at the same side of v_0, which makes g_1 work as a diamond gadget, and $v_0 \to v_1$ is also vertical. The same situation propagates through g_2 and g_3, and finally vertex pair p_2 is also vertical, which leads to contradiction. Thus, the 2.5-D cuboidal dual does not exist.

Otherwise if we have at least one 6-vertex gadget with $v_0 \to v_3$ horizontal, then we can place $v_0 \to v_1$ horizontal on this gadget, and the following gadget also has horizontal $v_0 \to v_4$. Regardless of the direction of $v_0 \to v_3$ on following gadgets, we can always make v_1 on the opposite side of v_4, i.e., $v_0 \to v_1$ horizontal. By this propagation, $v_0 \to v_1$ of g_3 is horizontal and the 2.5-D cuboidal dual of figure 2.16(a) can be constructed. □

With lemma 5, the reduction from Planar 3-SAT becomes straight forward, since in 3-SAT a clause is true if and only if at least one of its literals is true.

Theorem 3 *Planar 3-SAT reduces to 2.5-D cuboidal dual with 3 layers.*

PROOF. We construct a 3-layer graph $G = (V, E, L : V \mapsto \{1, 2, 3\})$ from $G_{p3SAT} = (U \cup C, E')$. G's overall topology is shown in figure 2.17. A pair of vertices on layer 2 defines the "vertical" direction, orthogonally connected with m pairs of vertices (like figure 2.15(a)) on layer 3, which are thus in horizontal direction. Each vertex pair on layer 3 is orthogonally connected with the lower pair of vertices in a clause gadget c_i (so that p_1 is vertical and p_2 is horizontal).

We create n diamond gadgets on layer 2 corresponding to variable $u_1, \cdots, u_n$. For each u_i, we put m pairs of vertices $p_{i1}, \cdots, p_{im}$ over u_i's diamond gadget on layer 1, which are all enforced in the same direction (as in figure 2.18(a)). Then for each u_i's appearance in clause c_j, or an edge in G_{p3SAT} between u_i and c_j, p_{ij} is connected to a path $R_{i,j}$ consisting of a series of layer 2 diamond gadgets interleaved with layer 1

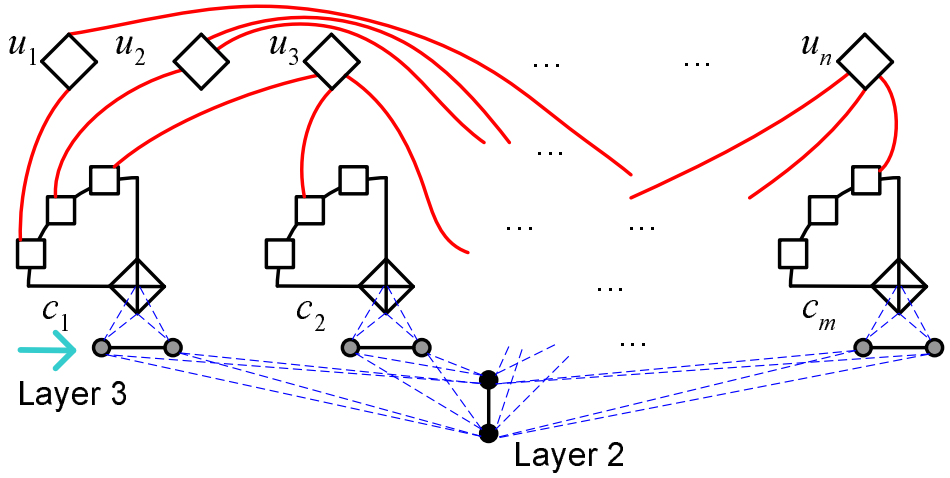

Figure 2.17: Construction of a 3-layer graph G from a Planar 3-SAT formula

vertex pairs, and the end of $R_{i,j}$ is connected to a 6-vertex gadget g_k in clause gadget c_j. $k \in \{1, 2, 3\}$ is determined by the rectilinear path embedding of G_{p3SAT} by lemma 4, such that the paths connected to g_1, g_2, g_3 are in clockwise order. For example, in figure 2.19's instance, clause gadget c_1 can have g_1, g_2, g_3 connected to the paths from u_1, u_3, u_2 respectively.

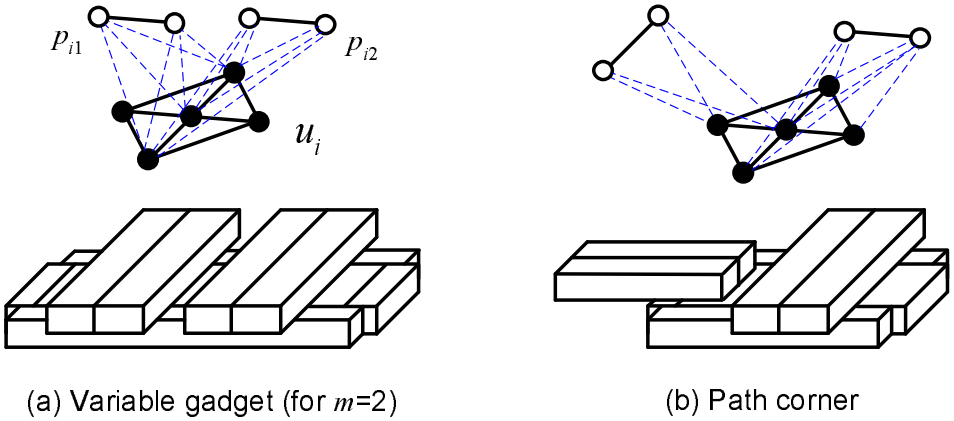

Figure 2.18: Variable gadget and path corner gadget

$R_{i,j}$ starts from vertex pair p_{ij} and ends at vertex pair (v_0, v_3) of one of c_j's 6-vertex gadget (g_k). Along $R_{i,j}$, the number of layer 2 diamond gadgets in $R_{i,j}$ is:

- $2(m+n)$, if u_i appears in c_j in non-negated form (as "u_i");
- $2(m+n)+1$, if u_i appears in c_j in negated form (as "$\overline{u_i}$").

A layer-2 diamond gadget connects two layer-1 vertex pairs as in figure 2.18(b). This connection both enables and enforces the two vertex pairs to have orthogonal directions,

so that a corner of a rectilinear path can be realized, regardless whether it is a left turn or right turn. Also,

- for u_i, vertex pair (v_0, v_3) in g_k is in the same direction of p_{i1};
- for $\overline{u_i}$, vertex pair (v_0, v_3) in g_k is in the orthogonal direction of p_{i1}.

Now we prove the Planar 3-SAT formula is satisfiable if and only if G has a cuboidal dual. The main part is from left to right, i.e., if there is a set of values assigned to $u_1, \cdots, u_n$ to make the formula true, then there is a 3-layer cuboidal dual of G, which can be construct from the set of u_i values.

The n variable gadgets and m clause gadgets are placed on the grid points of G_{p3SAT}'s rectilinear path embedding in lemma 4. For each variable u_i, its m pairs of vertices $p_1, \cdots, p_m$ are placed horizontal if $u_i = 1$, or vertical if $u_i = 0$. So by each path $R_{i,j}$ connected to (v_0, v_3) in g_k of clause gadget c_j, $v_0 \to v_3$ is horizontal if and only if u_i or its negation is evaluated to 1 in clause c_j. The clause gadgets can be constructed by lemma 5, because assuming the formula is evaluated to 1, each clause has at least one literal evaluated to 1, and the corresponding g_k has horizontal $v_0 \to v_3$.

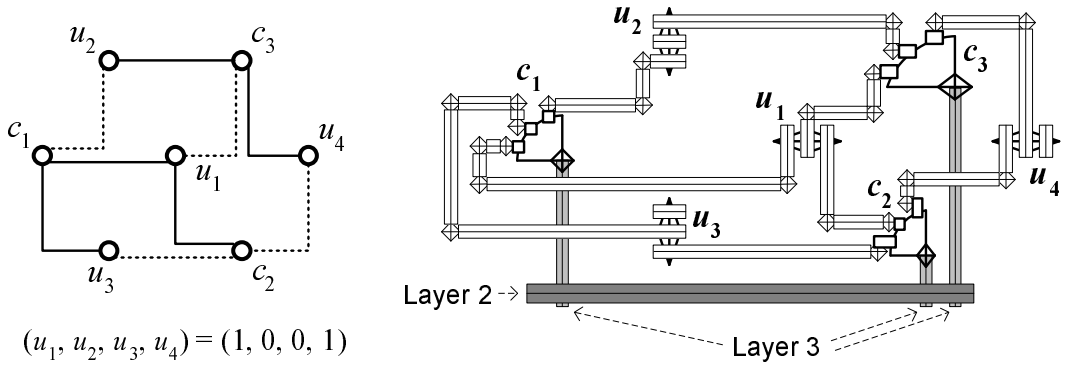

Figure 2.19: 3-layer cuboidal dual from 3-SAT formula $(u_1 \vee \overline{u_2} \vee u_3) \wedge (u_1 \vee \overline{u_3} \vee \overline{u_4}) \wedge (\overline{u_1} \vee u_2 \vee u_4)$

To make a complete 3-layer cuboidal dual, we add the layer-3 alignment gadgets and all the $R_{i,j}$ paths. As in the example of figure 2.19, the vertical layer-2 cuboid pair can be placed at bottom ($-y$ side), from which m horizontal layer-3 cuboid pairs connect to the m clause gadgets. By the shape of the clause gadget, the layer-3 pairs are placed at $+x$ side of the grid points; and if multiple clause gadgets exist on the same vertical line, we can make the upper gadgets wider so that the layer-3 pairs have no conflict. The layer-1 cuboid pairs of each path $R_{i,j}$ go along the path in the rectilinear path embedding, which needs no more than $2(m+n)-4$ corners by lemma 4. Consider that

g_1, g_2, g_3 are located at left or upper side each clause gadget c_j, this path may need 2 more corners to go around c_j. $R_{i,j}$ has at least $2(m+n)$ corner gadgets (figure 2.18(b)), which is enough to make the path, and redundant corners can be placed at any corner. (For convenience, in figure 2.19 we draw even number of corners for u_i and odd number of corners for $\overline{u_i}$.) Each corner gadget has a diamond gadget on layer 2, which is not contacting any layer-3 vertex. This is true at the two ends of each path, since the layer-3 vertex pairs are at $+x$ side of grid points, and the paths are on grid or at $-x$ side of grid points (by operation in lemma 4). And if a path segment in the middle overlaps with a layer-3 vertex pair, we move the path segment towards $-x$ and resolve the overlapping.

By the construction above, the 3-layer cuboidal dual is guaranteed to exist if provided an assignment of $u_1, \cdots, u_n$ satisfying the planar 3-SAT boolean formula. On the other hand, if there is a 3-layer cuboidal dual of graph G, then the 3-SAT formula is satisfiable, because by lemma 5 the orientations of the n variable gadgets give a solution set $u_1, \cdots, u_n$ that makes every clause evaluated to 1.

In conclusion, by constructing layered graph G from the 3-SAT instance, we reduce Planar 3-SAT to 2.5-D cuboidal dual problem with 3 layers. This is a polynomial reduction, because the number of vertices and edges in G is on $O(n^2)$ scale of G_{p3SAT}'s size or the 3-SAT formula's length. □

Corollary 3 *The problem of finding a layered graph's 2.5-D cuboidal dual is NP-complete if the number of layers in the graph* ≥ 3.

The hardness of finding multiple-layer cuboidal dual is also fundamentally greater than 2-D (single-layer) cases, except that the exact hardness of the 2-layer case is still unknown. Considering the gadgets introduced above are all in 2 layers, and the proof of NP-hardness only uses $2m$ vertices in the third layer, the 2-layer case of this problem may also be hard.

Acknowledgement

Chapter 2, in part, is a reprint of the paper "Complexity of 3-D Floorplans by Analysis on Graph Cuboidal Dual Hardness", co-authored with Evangeline Young and Chung-Kuan Cheng. To appear in *ACM Transactions on Design Automation of Electronic Systems*. The dissertation author is the primary investigator and author of this paper.

Chapter 3

Low Power Bus Architectures

3.1 Bus Architectures and Bus Gating

Standard on-chip buses like AMBA in figure 1.2 are designed to enable fast and convenient integration of system components into SoCs (system-on-chip), where simplicity is one of the major objectives. When the bus power consumption comes to a significant level that we cannot afford to ignore [LR04], power optimization will become a necessity. In this chapter, we introduce a physical bus matrix synthesis scheme based on a "bus gating" technique [WCSC09] to minimize the power on bus lines.

The main difference between a bus and a bus matrix is on the bandwidth capability. The original bus allows one master access at a time, and the bus matrix is extended to allow multiple accesses due to the demand of system parallelism. Simply making multiple copies of the basic bus is a possible way to realize a bus matrix, but the efficiency will be very low on both power, wires and other resources.

3.1.1 Power and wire efficiency of gated bus

The power efficiency of traditional bus architecture like figure 1.2 is low, because the bus lines from masters to slaves are connecting all the slave devices by a single large wire net. The same is on slave-to-master connections. While the communication is one-to-one, the signals are sent to all the receivers regardless whether they are needed, which results in wasted dynamic power on bus wires and component interfaces. Moreover, this low power efficiency is still being worsened by the technical scaling of global wires [HMH01] and the increasing number of components integrated in SoCs.

Gated bus is a solution to save the wasted dynamic power. The simplest way is to add a de-multiplexer (consisting of logic gates) after each multiplexer in figure 1.2, so that the signals only propagate to where they are needed. The idea works in a similar way as clock gating [DIBM03], and can be even more effective because the signal receivers here have much less complex behaviors than in a clock tree.

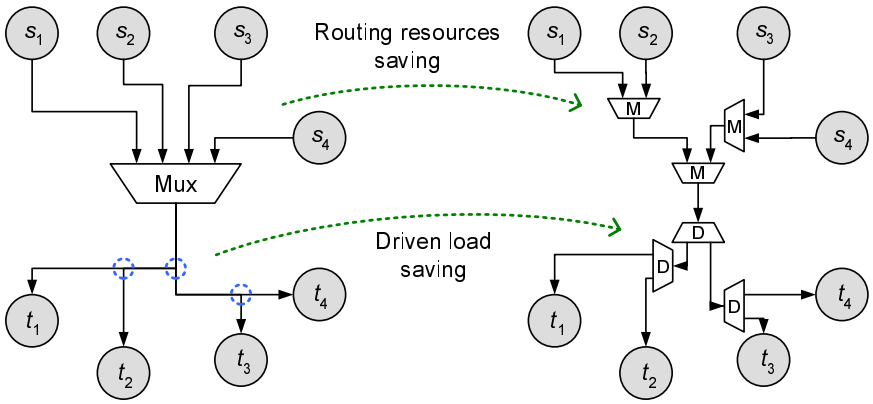

Figure 3.1: Bus gating using distributed mux and demux

We first look at tree structures, where the gating technique can be relatively simple. By distributing the multiplexer and de-multiplexer into the wire net as in figure 3.1, we can save both power and wires. For wire length, while the single multiplexer needs independent lines from every sender, the lines can be shared with distributed multiplexers and form a Steiner arborescence [CKL98] [RSHS92] [SC00]. An arborescence is a directed tree that every root-to-leaf path is shortest. On the receivers' side with distributed de-multiplexers, the bus lines change from a rectilinear Steiner minimum tree (RSMT) [GRSZ94] [HP94] to a minimum rectilinear Steiner arborescence (MRSA). By the research in [AKSW06], this change increases the wire length by only 2% $\sim$ 4% on average. So the total bus wire length can be reduced by the distributing the multiplexer/de-multiplexers, while the dynamic power can also be reduced at the same time. There is a small control overhead for sending the signals over the arborescence, but compared to the bus width and data throughput, this dynamic power overhead is negligible. Based on the same tree topology, effective bus gating can be applied by distributing the control over the entire tree (arborescence).

The Steiner arborescence in figure 3.1 provides a a tree structure such that every path from its root (between the last multiplexer and the first demultiplexer) to a leaf

takes the shortest path. However, the path length from source (e.g. s_1) to terminal (e.g. t_1) is still larger than the Manhattan distance between them. The tree structure has limitations in that there is only one root placed at a certain point, which may not be on the shortest path of every connection. For overall path optimizations, we use Steiner graph to replace the Steiner arborescence.

As defined in [BCE05], for an unweighted graph $G = (V, E)$, $G'' = (V'', E'', \omega)$ is a Steiner graph of G if $V \subseteq V''$ and for any pair of vertices $u, w \in V$, the distance between them in G'' is at least the distance between them in G. The example in figure 1.3 is a Steiner graph of G with $V = \{s_1, s_2, t_1, t_2\}$ and $E = \{(s_1, t_1), (s_1, t_2), (s_2, t_1), (s_2, t_2)\}$ with each edge weighted 1, and its implementation as a bus or bus matrix. This graph is minimal in terms of total wire length. Moreover, we call graph G'' a shortest-path Steiner graph of G, because every edge in E has a path in G'' with minimum length, i.e., the Manhattan distance between the two vertices. In this way, each data transaction involves minimal wires, leading to minimal dynamic power on bus lines.

Besides the power efficiency, shortest-path Steiner graphs also have advantages on bandwidth capability. The Steiner graph in figure 1.3 naturally supports both master devices accessing both slave devices at the same time, and therefore can be used as a 2×2 full bus matrix. As a result, graph structures are more suitable for physical synthesis of bus matrix. Our objective is to perform a balanced optimization on power and bandwidth even when available routing resource is limited.

3.1.2 Design flow with bus gating

The bus gating technique may bring some additional complexity in the design flow. Traditionally, physical level design starts from gate level netlist, and goes through placement, routing, timing analysis, verification, etc. With bus gating, the topology of the gated bus depends on floorplan and placement; and since the bus itself, with wires and control units, is usually included in the system, floorplan and placement also depend on bus connections, which form a closed loop of mutual dependency.

To resolve this loop, we can change the design flow by inserting the bus gating stage into placement and routing. After the initial placement considering the bus as a big wire net, the gated bus synthesis is performed, with a minimal update on the netlists and placement of bus units. Since the updates are limited to the bus or bus matrix part of the system, the process can be controlled in small scale and mostly automated. Provided

with appropriate algorithms, the impact of bus gating on design flows is minimal.

3.2 Shortest-Path Steiner Graph

We formulate our bus matrix synthesis scheme as a graph optimization problem. Given a source s and a set of terminals $V_t = \{t_1, t_2, \cdots, t_n\}$ placed on a plane with $P : V \mapsto R^2$, a minimal rectilinear Steiner arborescence (MRSA) is a minimal tree containing all the vertices, such that every path in the tree from s to a terminal t_i has the minimum length, which is Manhattan distance $\|P(s) - P(t_i)\|_1$. Extending from MRSA, given a set of sources V_s and a set of terminals V_t, a minimal shortest-path Steiner graph (MSPSG) is a minimal graph containing all the vertices, such that every path in the graph from a source s_i to a terminal t_j has length $\|P(s_i) - P(t_j)\|_1$.

The optimization flow is to first construct a shortest-path Steiner graph based on the given placement of $V_s \cup V_t$, and then decide the weight $\omega(e)$ on each edge to meet the bandwidth requirement. The graph construction algorithm is based on the minimum rectilinear Steiner arborescence (MRSA), which is well studied in previous work such as [CKL98] and [RSHS92]. Although it is proved to be NP-complete in [SC00], heuristic algorithms can provide close-to-optimal solutions of MRSA. Our shortest-path Steiner graph is formed by multiple iterations of a revised MRSA construction.

3.2.1 Heuristics for shortest-path Steiner graph

The RSA/G heuristic for the MRSA problem was first introduced in [RSHS92] and proved to be 2-approximate. Given a single source and n terminals, the basic flow is to start with n subtrees and iteratively merge a pair of subtree roots v and v' such that the merging point is as far from the source as possible, so that the wires can be shared as much as possible. It terminates when only one subtree remains. For efficient implementation, the RSA/G first sorts all the nodes on the Hanan grid [Zac00] with decreasing distance to the source, and visits each node while maintaining a peer set P of subtree roots. Details are shown in the pseudo code in table I, where two operations are used at: terminal merger opportunity (TMO), when a terminal is added into P as a subtree; and Steiner merger opportunity (SMO), when $|X| \geq 2$ and the subtrees in X are merged. In figure 3.2, the tree rooted at s_1 is an MRSA on s_1 and V_t.

The k-IDeA/G (iterated k-deletion for arborescence) algorithm is developed in [CKL98] based on the RSA/G. In each iteration, it removes up to k nodes from $v_1, \cdots, v_N$

Table 3.1: The RSA/G algorithm

Given a source s and n terminals $t_1, \cdots, t_n$,
$v_1, \cdots, v_N$ are the Hanan grid nodes of $\{s, t_1, \cdots, t_n\}$
sorted by decreasing distance to s
$Q \leftarrow \phi$;
for $i = 1$ to N do
if there is t_j at v_i, then (**TMO**)
$Q \leftarrow Q \bigcup \{v_i\}$;
$X \leftarrow Q \bigcap \{v_j : \|P(s) - P(v_j)\|_1 =$
$\|P(s) - P(v_i)\|_1 + \|P(v_i) - P(v_j)\|_1\}$;
if ($\|X\| \geq 2$) then (**SMO**)
merge the nodes in X rooted at v_i
$Q \leftarrow (Q \bigcap \overline{X}) \bigcup \{v_i\}$;
return the arborescence rooted at s;

when running the RSA/G algorithm. By removing the nodes, some SMO merges are skipped, which in some cases can result in a better overall solution. All the combinations of the k or fewer skipped nodes are tried in an iteration, and the best set of skipped nodes are marked as permanently deleted. The iterations are repeated until no further improvement occurs.

For the shortest-path Steiner graph with multiple sources $s_1, \cdots, s_m$, our algorithm actually constructs a graph H containing all the MRSAs from every source. While a single arborescence can be optimized by the k-IDeA heuristic, the set of m arborescences can be individually optimized with the same idea, plus that these arborescences also need to share as much wire as possible to optimize the final Steiner graph. For this purpose, we add additional heuristics based on the RSA/G to construct multiple MRSAs one by one.

First, starting from the second MRSA construction, we can reduce terminals by using existing wires. For each MRSA with source s_i, the terminals that need connection from the source can be moved along existing edges of H towards s_i. As the example shown in figure 3.2, with the wires of previous aborescences, we only need to connect 8 nodes instead of the original 16 terminals to form the MRSA rooted at s_2, because all the other terminals can be reached from one of these 8 nodes by a shortest path from s_2. This set of nodes (denoted as T') can be obtained by checking each terminal t_j, move from t_j towards s_i as much as possible along existing paths until reaching a vertex (can be a terminal or a Steiner node) in H where no vertex closer to s_i can be reached, and add this vertex to T'. When there are multiple paths in the graph, we pick the final

vertex closest to s_i, so the rest part of the path is short and likely to need shorter wires. Details are in the routine "Necessitate(v)" in table II.

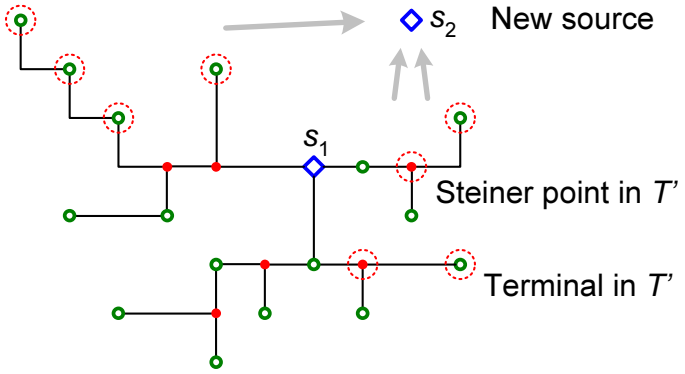

Figure 3.2: Nodes requiring connections (in dotted circles)

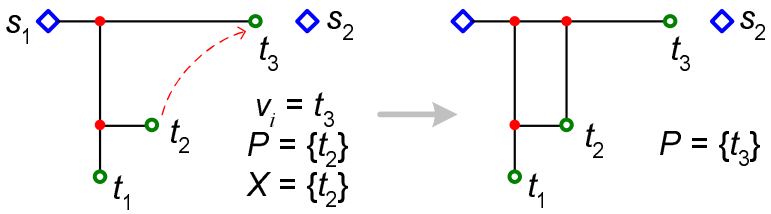

Figure 3.3: Connecting a node into the Steiner graph

Second, we construct the MRSA based on the set of nodes T' using as much existing wires as possible. Compared to the RSA/G heuristic, the TMO condition is changed to $v_i \in T'$; The SMO condition is changed, also for the purpose of wire reusing, from $|X| \geq 2$ to $|X| \geq 2$ or ($|X| = 1$ and $v_i \in H$). Because when v_i is already in the graph, it was added into previous MRSAs and can share wires with the node in X like the case in RSA/G when $|X| \geq 2$. As the example in figure 3.3 shows, when X contains only one node $\{t_2\}$, it should be connected into H when v_i comes to t_3, and half of the connection length can be saved using the existing horizontal wire. The detailed algorithm is described in table II, where the routine "connect(u, v)" uses existing wires if applicable on shortest connections.

The k-IDeA iterations remain unchanged here. And after the shortest-path Steiner graph is constructed by applying k-IDeA on the m sources, there are possibly some redundant edges that can be removed. So the final step is to check each edge $(v_i, v_j) \in H$, if H still contains all the source-to-terminal shortest paths without (v_i, v_j), then remove this edge from H.

Table 3.2: Revised RSA/G' algorithm

Given existing Steiner graph G, source s_k, terminals $t_1, \cdots, t_n$, and Hanan grid nodes $v_1, \cdots, v_N$ are same as in RSA/G;

Routine Necessitate(vertex v);
  $U \leftarrow \{u \in G$ and exists a wire path from v to u of
    length $\|P(s_k) - P(v)\|_1 - \|P(s_k) - P(u)\|_1\}$;
  $T' \leftarrow T' \bigcup \{u_m \in U$ with minimum $\|P(s_k) - P(u)\|_1\}$;

$T' \leftarrow \phi$;
for $i = 1$ to n do Necessitate(t_i);
$Q \leftarrow \phi$;
for $i = 1$ to N do
  if $v_i \in T'$ then $Q \leftarrow Q \bigcup \{v_i\}$; (**TMO**)
  $X \leftarrow Q \bigcap \{v_j : \|P(s_k) - P(v_j)\|_1 =$
    $\|P(s_k) - P(v_i)\|_1 + \|P(v_i) - P(v_j)\|_1\}$;
  if ($|X| \geq 1$ and $v_i \in G$) then (**SMO**)
    for each ($u \in X$) connect(v_i, u);
    $Q \leftarrow Q \bigcap \overline{X}$;
    Necessitate(v_i);
  else if ($|X| \geq 2$) then (**SMO**)
    merge the nodes in X rooted at v_i
    $Q \leftarrow (Q \bigcap \overline{X}) \bigcup \{v_i\}$;
return; (the MRSA rooted at s_k is added to G)

Examples of shortest-path Steiner graphs constructed from this algorithm are shown in figure 3.4, 3.6 and 3.10.

3.2.2 Edge weights by max-matching

With graph topology constructed, we have at least one shortest path for every master-to-slave connection. For the purpose of simplicity and small foot print of bus control units, each connection takes a fixed path in the Steiner graph. And from all the paths, we can decide the weight on each edge, i.e., number of parallel bus lines along each edge. For instance, figure 3.4 shows the shortest-path Steiner graph constructed on 3 sources and 5 terminals. There are totally $3 \times 5 = 15$ pairs of connections, and all the 15 paths are also drawn in the figure. With this set of paths, the bus lines drawn on the Steiner graph can handle any set of communications, as long as each master or slave device is communicating to at most one other device.

The weight $\omega(e)$ on each edge e in the Steiner graph can be decided from the bandwidth requirement and connection paths. We require maximum bandwidth capability for the bus matrix, so that a master device can access any idling slave through the

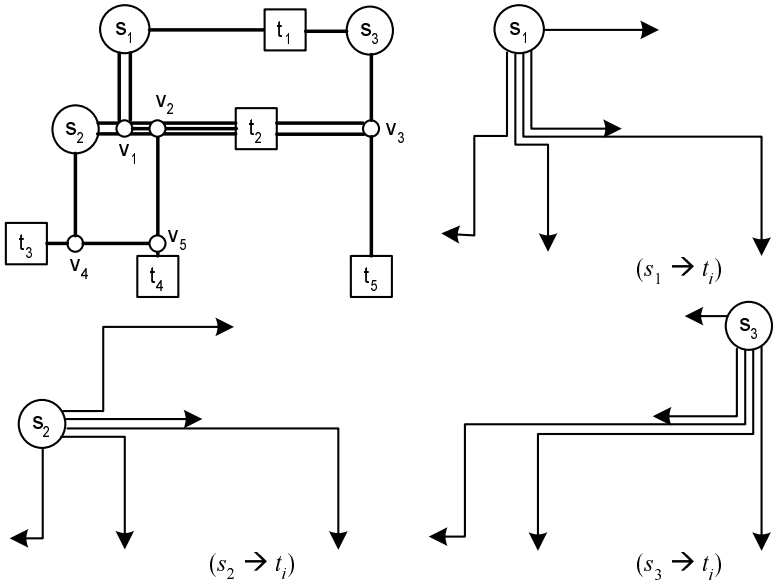

Figure 3.4: A bus matrix graph and all its master-slave connections

bus matrix. This means the communication bandwidth is only limited by the devices interface (one connection at a time), not limited by the bus matrix connecting the devices. Under this assumption, $\omega(e)$ is determined by the maximum subset of paths going through edge e, such that no paths in the subset share a source or a terminal. This maximum subset is exactly the maximum matching on a bipartite graph $G' = (V_s, V_t, E')$, where $(s_i, t_j) \in E'$ when the connection path from s_i to t_j goes through edge e.

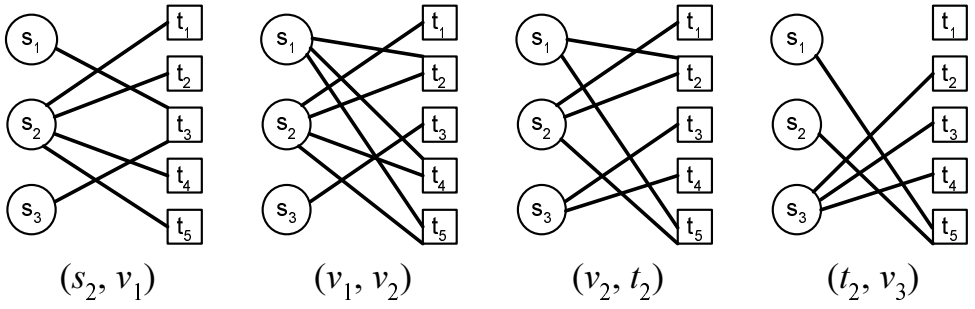

Figure 3.5: Bipartite graphs of 4 edges in the bus matrix of figure 3.4

In this way, wires can be shared by connections that do not happen at the same time. For the shortest-path Steiner graph in figuer 3.4, the bipartite graphs of edges (s_2, v_1), (v_1, v_2), (v_2, t_2) and (t_2, v_3) are shown in figure 3.5. The upper limit of edge weight is 3, since there are 3 master devices. Nevertheless, most edges in the Steiner

graph have lower weight than the limit, especially those located at non-central areas.

3.3 Tradeoffs on Power and Wires

The shortest-path Steiner graph constructed above is optimal for minimal power on bus wires, without minimizing area and routing resources. The area overhead can be very small because of the simple control mechanism, but routing resource may become a bottleneck depending on other design factors. Therefore, we need to explore some tradeoffs among the objectives in order to have more flexible choices.

By observing figure 3.6(a), there are some long and narrow rectangles formed by the graph edges, and the long double edges are necessary if we want all the connections to be shortest. But when the double edges are geometrically very close to each other, combining them into one edge only slightly increases the length of some connections, while possibly saving much more wire length. Figure 3.6(b) and 3.6(c) show the effect of merging parallel lines in narrow rectangles. The total edge length is greatly reduced, while the increment on average path length is relatively small. Although fewer edges will generally result in larger edge weight, the total weighted edge length (wire length) can still be reduced by this merging operation due to improved wire sharing among paths.

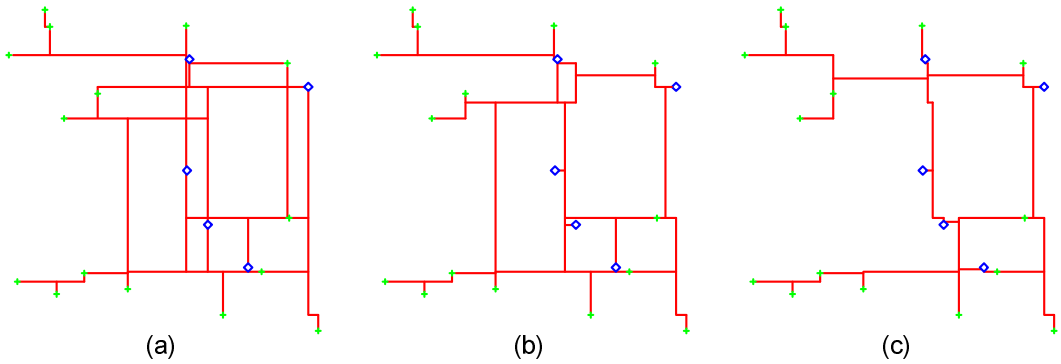

Figure 3.6: Merging parallel lines in a Steiner graph

Thus, if we relax the requirement on the path length in definition 4, from the exact Manhattan distance $\|P(u)-P(v)\|_1$ to within $(1+\epsilon)\|P(u)-P(v)\|_1$, we can merge the double parallel lines to save wires. Assume we have a vertical narrow rectangle with dimensions $h \times w$, and we merge the two vertical edges to a single edge placed in middle. The total edge length can be reduced by h, while the lengths of some connection paths increase by $\frac{w}{2} + \frac{w}{2} = w$. So if the h/w ratio is high, this operation of merging parallel

lines can be very helpful on relieving routing congestion, while preserving the low power consumption of a bus matrix.

In the wire length reduction algorithm, we repeatedly search for pairs of parallel double lines in the bus matrix graph, and for each pair, calculate its potential reduction Δl on edge length and possible increment Δp on path lengths. The pair with highest $\Delta l/\Delta p$ ration is merged, and the modified graph will have a new set of connection paths and edge weights. If the added total wire length is really reduced, we keep the merging operation and continue to the next iteration, otherwise discard the operation. Eventually, there will be no positive wire length reduction in the graph, and we will have a series of bus matrix graphs with decreasing wire length and increasing path lengths, where a comprise can be chosen.

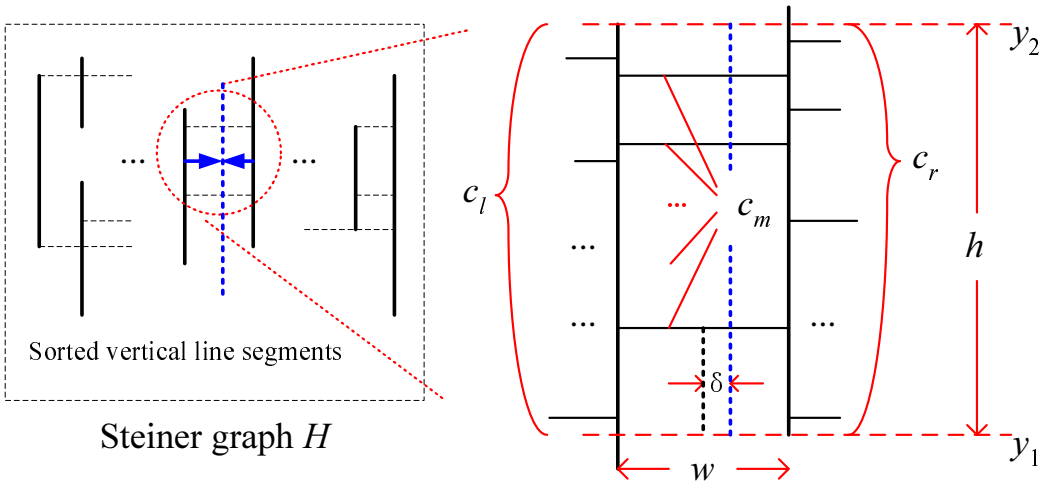

Figure 3.7: Searching for mergeable parallel segments (in vertical direction)

The process of searching for vertical mergeable parallel line segments is illustrated in figure 3.7. (Horizontal lines are processed in the same way with x-y coordinates switched.) First, the vertical line segments in the Steiner graphs are sorted by their x coordinates, denoted as $u_1, u_2, \cdots, u_k$. Then for each pair of segments u_i, u_j $(i < j)$ with a common y interval $[y_1, y_2]$, if between i and j there is no other vertical segment on $[y_1, y_2]$, u_i and u_j are a pair of mergeable segments.

On the parallel segments u_i and u_j, let c_l denote the count of horizontal lines connected to the left, c_r denote the count of lines connected to the right, and c_m the count of lines connecting u_i and u_j in the middle. Assume $c_l < c_r$, so the combined vertical segment may have an offset δ to the right of the midpoint.

The reduction on total edge length Δl is by combining the vertical segments of length h and changing the lengths of horizontal connections, so $\Delta l = h + c_m w - c_l(\frac{w}{2} + \delta) - c_r(\frac{w}{2} - \delta)$. On the possible increment on path lengths, since the left vertical segment is pushed rightward by $\frac{w}{2} + \delta$, a path may need to detour and add $\Delta p = w + 2\delta$ of distance. So the ratio is

$$\frac{\Delta l}{\Delta p} = \frac{h + c_m w - c_l(\frac{w}{2} + \delta) - c_r(\frac{w}{2} - \delta)}{w + 2\delta}$$

$$= \frac{c_r - c_l}{2} + \frac{h - (c_r - c_m)w}{w + 2\delta}$$

The best offset value δ can be decided by the right part $\frac{h-(c_r-c_m)w}{w+2\delta}$ to maximize the ratio. If the upper part $h - (c_r - c_m)w \geq 0$, i.e., $\frac{h}{w} \geq c_r - c_m$, then let $\delta = 0$ so that the merged vertical segment is placed at middle. Otherwise $\frac{h}{w} < c_r - c_m$, let $\delta = w/2$ which is the maximum offset value, and the merged segment is at the right segment u_j's position (note we assume $c_l < c_r$, otherwise δ is negative).

In figure 3.6, the graphs are the stages of the merging iterations applied on a Steiner graph. First, the long and narrow rectangles are removed, followed by wider rectangles. If we do not require high bandwidth capability, i.e., edge weights all set to 1, the final graph has about half total edge length of the original shortest-path Steiner graph. On weighted graphs of bus matrices, the reduction on total wire length is usually smaller, since the number of connections is unchanged, and the edge weights are increased by the merging operations. Nevertheless, we still can achieve a significant reduction on total wire length. More test cases and data are shown in the last section.

Combining the algorithms of graph generation and edge merging, the overall optimization strategy is shown in table 3.3. Based on the shortest-path Steiner graph which achieves minimal power, this flow can produce a series of graphs with decreasing total wire length and increasing average path length, each one optimized according to device locations and communication patterns. With real world power budget and resource availability, designers can choose a best compromise between power and wires.

3.4 Estimations on Control Overhead

Apart from path lengths and data wire lengths, the control overhead needs to be considered for a complete optimization. Although the data lines consume the major amount of routing resource since they are usually at least 64-bit (32-bit $\times$ 2-way) wide,

Table 3.3: Bus matrix physical synthesis

Given a set of sources $s_1, \cdots, s_n$, a set of terminals $t_1, \cdots, t_n$, and a location function $P : S \bigcup T \mapsto R^2$
Routine **Set_Edge_Weights**(Steiner graph H) For each connection (s_i, t_j), find a shortest path $R_{i,j}$ in H from s_i to t_j; For each edge $e \in E$ in H, $A' \leftarrow \{(i,j) : e \in R_{i,j}\}$; $\omega(e) \leftarrow$ Max_matching(A');
1. Generate shortest-path Steiner graph $H_0 = (V, E)$; (by Table 3.2) 2. **Set_Edge_Weights**(H_0); 3. $i \leftarrow 0$; 4. Repeat 4.1 Find all pairs of parallel segments in H_i, and sort them in stack $D[]$ by decreasing $\Delta l/\Delta p$; 4.2 While ($D[]$ is not empty) $(u_i, u_j) \leftarrow$ Pop out the segment pair in $D[]$; $H_{temp} \leftarrow H_i$ with segment u_i and u_j merged; **Set_Edge_Weights**(H_{temp}); If ($H_{temp} < H_i$ on total wire length) $i \leftarrow i + 1$; $H_i \leftarrow H_{temp}$; Break the "while" loop; Until (no wire length reduction found in H_i) 5. Evaluate the bus matrix graphs $H_0, H_1, \cdots$ by design objectives

control overhead is increased compared to traditional bus architectures by adopting Steiner graphs. We need a lot of switches at Steiner nodes to guide the on-chip traffic, and each switch needs a certain number of control signals depending on its node degree and the weight on each of the edges.

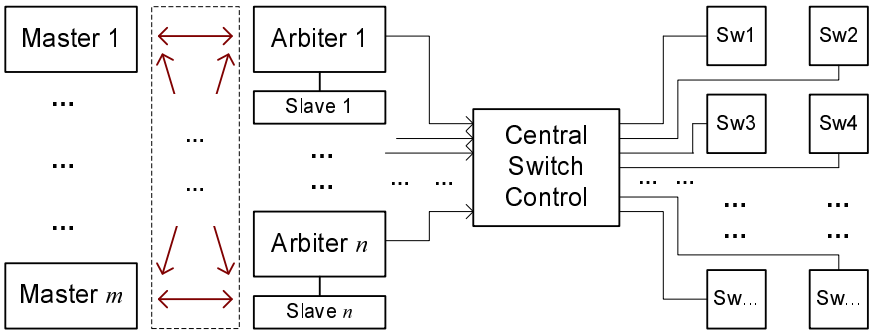

Figure 3.8: Control on switches in a bus matrix

The sketch of bus matrix control scheme is shown in figure 3.8. Each slave device has an arbiter which handles the requests from masters and decides the connection. The result is sent to the central switch control unit, where all the connection paths are stored. Depending on the set of active paths, the central switch control sends control signals to all the switches on each path, which together instantly create the master-to-slave connection requested by the master device.

On wire length overhead, the control wires are from the central switch control to all the switches. A bus switch is basically a crossbar plus an auxiliary local control which remembers each path going through. The local control handles two types of requests from the central switch control, *create_connection(port1, port2)* and *dispose_connection(port1, port2)*. Typically, a Steiner node in the bus matrix graph has degree 3, and the combinations of (*port1, port2*) is $\binom{3}{2} = 6$, which can be distinguished by 3 control signals. Plus another signal for the create/dispose request, a 3-way switch needs 4 wires connected to the central switch control. For each slave device, there is an arbiter deciding which master has the access, so we have $\lceil \log_2 m \rceil + 1$ control signals from each arbiter to the central switch control. With a given placement of system components and control units, our algorithm generates the bus matrix graph with all the switches, and the total length of control wires can then be calculated. Compared to a typical bus width of 64 bits, the wire overhead for switch control is relatively low. More details are illustrated in table 3.4 and 3.5 in the next section.

On power overhead, the overhead on the control wires can be ignored. Because during a data transaction, the control signals do not toggle, while the data signals typically transmit a large amount of data and consume most of the power. Plus that the number of control wires is much smaller, the power percentage on control signals has no impact on the total bus power. Therefore, we only estimate the power overhead coming from the switches on Steiner nodes.

As constructed by the algorithm in table 3.2 and 3.3, the master-to-slave connections are all along the shortest or near-shortest paths in the Steiner graph. To implement the bus matrix, we need to put switches on Steiner nodes to guide all the connections. Compared to traditional bus architectures where the connections are by long lines inserted with buffers, our Steiner graph structure shortens the path length, but also adds crossbar switches which consume more power than basic buffers.

We use a crossbar design illustrated in figure 3.9. In this example, we have a

4-way switch at a junction of $N_n + N_w + N_s + N_e$ bus lines. It works like a miniaturized bus matrix, enabling 2-way connections between any pair of ports (except the pairs at the same side which never need connections). Figure 3.9 shows the wires connected to the multiplexer and demultiplexer of port 1 on the west side, where each "M" (mux) or "D" (demux) has $N_n + N_s + N_e$ connections and can be realized by a binary tree consisting of 2-to-1 mux/demuxes distributed in $\lceil \log_2(N_n + N_s + N_e) \rceil$ levels.

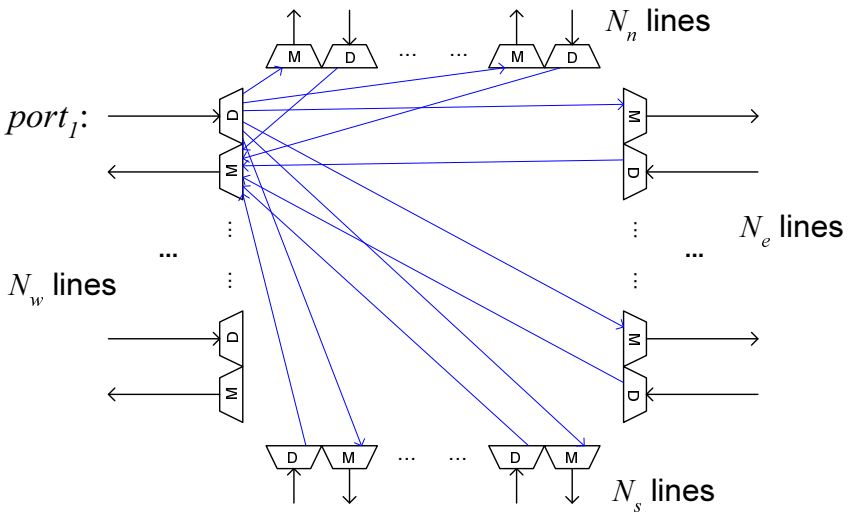

Figure 3.9: Implementation of a crossbar switch

Using this crossbar constructed by 2-to-1 mux/demuxes and wires, a path going through a switch from port_i to port_j is going through $\lceil \log_2(N - N_i) \rceil + \lceil \log_2(N - N_j) \rceil$ muxes and demuxes, where N is the total edge weight at the Steiner node and N_i, N_j are the corresponding edge weights. Hence, the power overhead is generally low because of the logarithmic scale on the number of mux/demuxes. Due to the scaling down of feature size, this overhead can be further reduced to an insignificant level. Also, more details are illustrated in the next section.

3.5 Experimental Results

In our experiments, we implement all the related algorithms, including shortest-path Steiner graph generation, Steiner graph reduction by parallel line merging, and edge weight maximum matching. The programs are tested on Windows Vista platform with a 2.2GHz Intel Core2 processor. The running time is short on all the test cases, because the algorithms are time/space efficient, and also because most SoC bus matrices

will not need to connect too many devices (under 100 in our cases).

The test cases we use are mostly artificial, hand made (T_0 and T_1) or randomly generated ($T_{2\sim12}$). They are the same cases used in [WCSC09] and [WYGC10]. In each test case, the master and slave devices are distributed over a 10mm×10mm square.

The power consumption is estimated by the driven capacitance of data transactions, therefore can be calculated as a linear combination of path length and switches along the path. According to the optimization in [ZCY+07], we estimate that under 90nm technology, each mux or demux in crossbar switches has about the same capacitance as 25μm of wires. So by using the switch design of figure 12, the dynamic power overhead of going through a switch (port_i – port_j) has the power overhead equivalent to adding $25(\lceil\log_2(N-N_i)\rceil + \lceil\log_2(N-N_j)\rceil)\mu$m of wires.

The total wire length on data wires and control overhead are added straightforwardly. Data wire length is the sum of weighted edge length in the bus matrix graph. The control overhead is estimated from figure 11, where the central switch control is placed at the center of the chip, and the control wires consist of those from slave devices to central switch control, and those from central switch control to all switches. We assume the data lines are 64-bit wide, and therefore the percentage on control wires are divided by 64.

We list the bus matrix synthesis results on all the test cases in table 3.4 and 3.5. Case $\text{T}_i(m,n)$ contains m master devices and n slave devices. Maximum bandwidth is

Table 3.4: Power and wire length results for minimal power

Case(m,n)	$\sum L_{v_s,v_t}$	$\overline{L}_{tree}$	$\overline{L}_{path}$	$\overline{P}_{switch}$	$\sum L_{wire}$	$\sum W_{ctrl}$
T_0 (3, 16)	315000	84180	6354	5.83%	93000	9.75%
T_1 (3, 16)	842000	145500	6912	8.52%	105000	9.23%
T_2 (2, 30)	426540	123050	6687	7.31%	101170	15.68%
T_3 (3, 16)	331850	82300	6912	8.52%	71680	13.04%
T_4 (5, 15)	563400	96740	6888	11.83%	141360	10.51%
T_5 (6, 16)	694940	112100	6940	12.34%	230380	9.57%
T_6 (8, 8)	482340	89520	6887	10.97%	146060	7.59%
T_7 (12, 6)	481540	121070	6567	10.40%	157020	7.25%
T_8 (16, 10)	1166020	14377	6830	18.92%	324290	6.65%
T_9 (8, 16)	831080	116520	6180	16.52%	272740	8.53%
T_{10} (8, 16)	1017020	128990	7487	15.66%	276630	8.48%
T_{11} (6, 12)	484400	89700	6685	11.51%	142650	8.78%
T_{12} (12, 12)	1008320	132420	6686	15.85%	274970	7.10%

Table 3.5: Power and wire length results for minimal wire length

Case(m, n)	$\sum L_{v_s,v_t}$	$\overline{L}_{tree}$	$\overline{L}_{path}$	$\overline{P}_{switch}$	$\sum L_{wire}$	$\sum W_{ctrl}$
T_0 (3, 16)	315000	84180	7850	6.95%	62000	10.39%
T_1 (3, 16)	842000	145500	6912	8.52%	105000	9.23%
T_2 (2, 30)	426540	123050	7519	11.70%	85850	19.35%
T_3 (3, 16)	331850	82300	8093	9.98%	69850	12.94%
T_4 (5, 15)	563400	96740	7232	16.65%	101080	13.83%
T_5 (6, 16)	694940	112100	7234	16.02%	174600	11.22%
T_6 (8, 8)	482340	89520	7933	12.15%	113240	9.84%
T_7 (12, 6)	481540	121070	7100	14.95%	116770	10.57%
T_8 (16, 10)	1166020	14377	7838	24.41%	207740	9.75%
T_9 (8, 16)	831080	116520	6646	21.78%	187180	11.00%
T_{10} (8, 16)	1017020	128990	8678	27.90%	163740	12.27%
T_{11} (6, 12)	484400	89700	6728	11.57%	129420	10.12%
T_{12} (12, 12)	1008320	132420	7299	22.55%	186800	11.83%

always required as assumed in section 3.2, i.e., a master device can always access any idle slave device without being limited by the number of data lines in the bus matrix. The objective can be minimum power, minimum wire length, or a combination of the two. Table 3.3 shows the results for minimal power and table 3.4 for minimal wire length.

At the top of each column,

- $\sum L_{v_s,v_t}$ is the sum of Manhattan distances on all the master-slave pairs;
- $\overline{L}_{path}$ is the average path length of master-slave connections in the bus matrix;
- $\overline{P}_{switch}$ is the added percentage of power overhead in data transactions by the switches on Steiner nodes;
- $\sum L_{wire}$ is the total data wire length;
- $\sum W_{ctrl}$ is the added percentage of control wire overhead.

In the minimal power table, the average path length is exactly $\sum L_{v_s,v_t}/(mn)$, while the total wire length is about one fourth to one third of the total connection length. Compared to traditional bus implementation in [WCSC09], the dynamic power saving is mostly over 90% even with the switching overhead added. Overhead on dynamic power increases with the number of components increasing, which requires more bandwidth and larger switches. The percentage is generally under 20% on random cases with under 30 components. So the overall dynamic power here is close to optimal. On the overhead of control wires, the percentage is mostly under 10%, because the number of control signals required is usually very low compared to the 64-bit wide data lines.

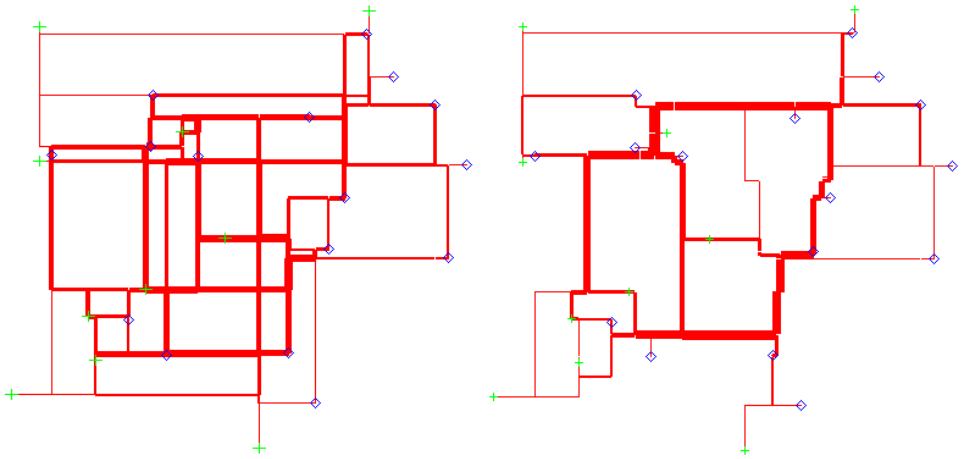

Figure 3.10: Case T_8's data wires under minimal power and minimal wire

In the minimal wire table, the Steiner graphs are reduced by the parallel line merging heuristic. As a result, the wire length on most cases is greatly reduced, except for the highly regular case T_1, which is already wire-efficient under minimal power. The data wire length reduction is around 30%, depending on the location distribution of components. And despite the slightly increased percentage, the control wire length is actually reduced, because there are less switches in the reduced bus matrix graph. Figure 3.10 shows the data wires in case T_8 under minimal power and minimal wire optimizations. The thickness of each segment indicates the number of repeated bus lines (edge weight $\omega(e)$).

Compared to the reduced wire length, the increase on average path length is much lower, mostly around 10% and all under 20%. The power overhead percentage is also increased, because although the Steiner nodes are reduced, the switches along each path are not reduced as much in number, but increased in size. Still, these solutions are relatively power efficient, and we have series of intermediate solutions between minimal power and minimal wire are available for choice.

The parallel segment merging heuristic (table 3.3) can provide a series of bus matrix graphs with decreasing wire length and increasing path length. Figure 3.11 provides detailed curves in case T_8 and T_{12}. Generally, the paths' length increase as wires being reduced except at a few points. According to power and wire budgets, designers can choose a point on the curve for the best compromise.

If allowed by the system performance requirement, bandwidth capability can be added into the tradeoff. With a bipartite communication graph, the bus matrix

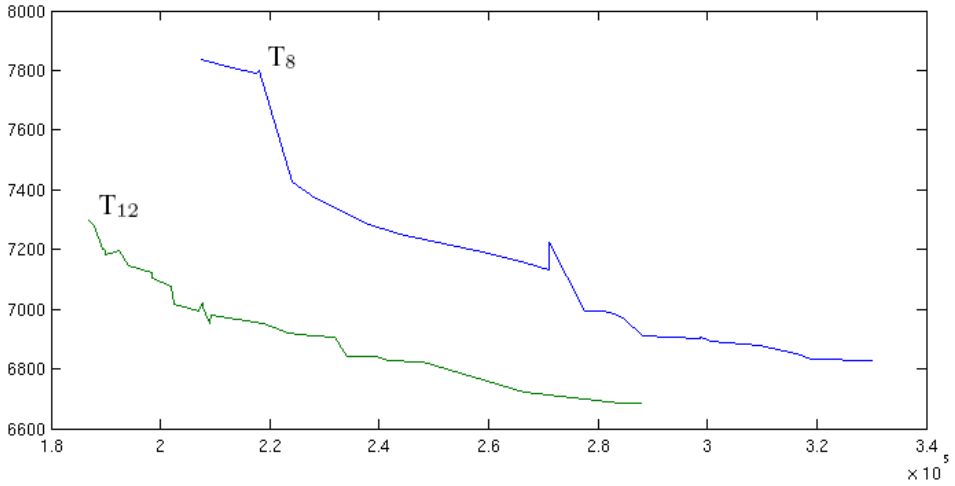

Figure 3.11: Average path length vs data wire length in case T_8 and T_{12}

should have the capacity of handling $\min(m, n)$ data transactions simultaneously. By reducing this capacity, wires can be saved, i.e., the tradeoff curve is pushed towards lower-left. For bandwidth capacity k, an edge e will need $\min(\omega(e), k)$ copies of bus lines. Applying to our test cases, the wire length start to drop when k goes below 5. But when $k > 5$, the wire reduction is very low. The reason is that the connection paths are randomly distributed over the graph, so very few edges have $\omega(e)$ close to the maximum bandwidth capacity $\min(m, n)$. Moreover, provided with detailed system communication patterns, maximum bandwidth capacity may not be required. Instead of assuming maximum bandwidth requirement, we can have a series of communications sets $C_1, C_2, \cdots, C_p$, each one being a subset of $\{(s_i, t_j) : 1 \leq i \leq m, 1 \leq j \leq n\}$, denoting a set of simultaneous connections. Using these sets in the max-matching algorithm of table 3.3, the edge weights can possibly be further reduced. In this case, more information on system-level behavior can provide extra space for optimizations.

Acknowledgement

Chapter 3, in part, is a reprint of the paper "Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications", co-authored with Yulei Zhang, Nan-Chi Chou, Evangeline Young, Chung-Kuan Cheng, Ronald Graham. Invited submission to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. The dissertation author is the primary investigator and author of this paper.

Chapter 4

Conclusions and Future Work

As power dominates circuit performance, and interconnect dominates the power, we look at two problems related to circuit and system interconnects to push forward through the "power wall". One is on the floorplanning stage, which is placing function modules in a 3-D space to reduce interconnect length; another one is on the bus synthesis stage, which is connecting the modules together to reduce interconnect wires involved in communication actions.

The cuboidal dual problem has 3 variations in 2-D, 2.5-D and 3-D. Naturally, the difficulty increases with the number of dimensions. A surprising finding among these results is that just a few layers of the 2-D cases, which can be decided by a simple rule (Theorem 2 or [KK84]), being stacked together, can make the problem so much more complex that there exists no effective algorithm to decide the solution (unless P = NP). The relation between topological connections and geometrical contacts in 2-D floorplans is not inherited when extended to 3-D structures. With the much increased complexity in 3-D structures, we may expect a big challenge for both designers and CAD tool developers in future 3-D IC design. Human intelligence will play a more important role in the design flow and in devising heuristic algorithms in 3-D floorplanning, placement and routing tools. More research in this cuboidal dual problem or other problems with graph-geometry formulations can be helpful for us to understand the nature of 3-D physical design problems.

In the bus matrix synthesis stage, interconnect lengths are already determined by module placement on the chip. The principle of our work on reducing power is to minimize the data movement on the chip; and that on reducing wires is to maximize

wire sharing among different connections. The methods and results have some similarities with city traffic planning and road construction. Like our fixed paths, most people in a city have a fixed route between work and home, and roads are constructed with various number of lanes (width) depending on local traffic density. The two graphs in figure 3.10 look like roads on a map, which may not be a coincidence but a result of similar principles and approaches. The overall result shows promising potentials of bus matrices applied for low power and high performance on-chip communications.

Future work includes, but is not limited to, a bus synthesis scheme extended to 3-dimensional circuits. Geometrically, the Steiner graph construction can be generalized into 3-D in a straightforward way. However, due to current manufacturing technologies, the wires on the third dimension need to be made as through-silicon-vias (TSVs), which are thicker than normal wires by orders of magnitude. These third dimension TSVs may consume a large amount of chip area and routing resources, and therefore greatly change the entire optimization problem. Possibly we will need new floorplanning and placement schemes with early stage considerations on efficient communication solutions. And as proved in chapter 2, the floorplanning stage is also gaining complexity, requiring higher level of design capability. As technology being pushed to the limits, more uncertainties and challenges await efforts of research and investigation.

Bibliography

[AKSW06] C. J. Alpertt, A. B. Kahng, C. N. Szet, and Q. Wang. Timing-driven Steiner trees are (practically) free. *ACM/IEEE Design Automation Conf.*, pages 389–392, 2006.

[BCE05] B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Math.*, pages 1029–1055, 2005.

[BS86] J. Bhasker and S. Sahni. A linear algorithm to find a rectangular dual of a planar triangulated graph. *ACM/IEEE Design Automation Conf.*, pages 108 – 114, 1986.

[CKL98] J. Cong, A. B. Kahng, and K.-S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. *IEEE Trans. Computer-Aided Design*, 17(1):24–39, Jan. 1998.

[CNAO85] N. Chiba, T. Nishizeki, A. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *J. Computer and Systems Sciences, 30*, pages 54–76, 1985.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. *3rd ACM Symp. Theory of Computing*, pages 151 – 158, 1971.

[DIBM03] M. Donno, A. Ivaldi, L. Benini, and E. Macii. Clock-tree power optimization based on RTL clock-gating. *ACM/IEEE Design Automation Conf.*, pages 622–627, 2003.

[DT01] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection network. *ACM/IEEE Design Automation Conf.*, 2001.

[GRSZ94] J. Griffith, G. Robins, J. Salowe, and T. Zhang. Closing the gap: Near-optimal Steiner trees in polynomial time. *IEEE Trans. Computer-Aided Design*, 13:1351–1365, 1994.

[HMH01] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings IEEE*, 89:490–504, 2001.

[HP94] C.-T. Hsieh and M. Pedram. An edge-based heuristic for Steiner routing. *IEEE Trans. Computer-Aided Design*, 13(12):1563–1568, Dec. 1994.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations. New York: Plenum.*, pages 85–103, 1972.

[KK84] K. Kozminski and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. *ACM/IEEE Design Automation Conf.*, pages 655–656, 1984.

[Lic84] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput. Volume 11, Issue 2*, pages 329–343, 1984.

[LR04] K. Lahiri and A. Raghunathan. Power analysis of system-level on-chip communication architectures. *Int'l Conf. Hardware-Software Codesign and System Synthesis*, pages 236–241, 2004.

[Moo65] Gorden Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[RSHS92] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[SC00] W. Shi and S. Chen. The rectilinear Steiner arborescence problem is NP-complete. *ACM-SIAM Symp. on Discrete Algorithms*, pages 780–787, 2000.

[Sch90] W. Schnyder. Embedding planar graphs on the grid. *ACM-SIAM Symp. on Discrete Algorithms*, pages 138–148, 1990.

[WCSC09] Renshen Wang, Nan-Chi Chou, Bill Salefski, and Chung-Kuan Cheng. Low power gated bus synthesis using shortest-path Steiner graph for system-on-chip communications. *ACM/IEEE Design Automation Conf.*, pages 166–171, 2009.

[WYGC10] Renshen Wang, Evangeline Young, Ronald Graham, and Chung-Kuan Cheng. Physical synthesis of bus matrix for high bandwidth low power on-chip communications. *ACM Int'l Symp. Physical Design*, 2010.

[YHH06] H. Yu, J. Ho, and L. He. Simultaneous power and thermal integrity driven via stapling in 3D ICs. *IEEE/ACM Int'l Conf. Computer-Aided Design*, pages 802 – 808, 2006.

[YSNK00] H. Yamazaki, K. Sakanushi, S. Nakatake, and Y. Kajitani. The 3d-packing by meta data structure and packing heuristics. *IEICE Trans. Fundamentals*, pages 639–645, Apr. 2000.

[Zac00] M. Zachariasen. A catalog of Hanan grid problems. *Networks*, 38:200–1, 2000.

[ZCY+07] Ling Zhang, Hongyu Chen, Bo Yao, Kevin Hamilton, and Chung-Kuan Cheng. Repeated on-chip interconnect analysis and evaluation of delay, power, and bandwidth metrics under different design goals. *Int'l Symp. Quality Electronic Design*, pages 251–256, 2007.