Technische Universität München
Lehrstuhl für Entwurfsautomatisierung

Design Automation for Continuous-Flow Microfluidic Biochips

Tsun-Ming Tseng

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technische Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. sc. techn. Gerhard Kramer
Prüfer der Dissertation:	1. Prof. Dr.-Ing. Ulf Schlichtmann
	2. Prof. Tsung-Yi Ho, Ph.D.

Die Dissertation wurde am 19.01.2017 bei der Technische Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 03.08.2017 angenommen.

Abstract

Continuous-flow microfluidics has evolved very rapidly in the last twenty years. Biochemical applications can be performed in parallel and automatically on continuous-flow microfluidic chips, by which more precise results with higher throughput can be achieved. To date, most continuous-flow microfluidics are still designed manually, which is time-consuming, error-prone, and especially impractical for large-scale-integration.

Design automation researchers started to develop design automation tools for continuous-flow microfluidics about ten years ago. Some of these are front-end tools, which scheduled biological operations from assay protocols and allocated these operations to microfluidic devices. Some others are back-end tools, which performed automatic device placement and micro-channel routing. As new on-chip components, e.g., sieve valves, and new architectures, e.g., a homogeneous valve-centered architecture, are continuously invented and adopted, design automation tools also evolve accordingly over time.

This dissertation first provides a brief review of design automation research for continuous-flow microfluidics over the last decade. In this review, a detailed description is given for the microfluidic architecture and the general topics of front-end as well as back-end research in design automation field. Selected research results are then introduced according to their appearance over time, which also matches the general trend of research topics from front-end to back-end. The main body of the dissertation presents new research results that cover four topics in four chapters: temporary caching of fluids, sieve valve exploration, synthesis for reconfigurable chips, and the layout generation tool Columba.

Contents

dedicated to my parents, Hsi-Kuo Tseng and Pao-Chin Huang:
thank you for supporting me to pursue my dream;
and dedicated to Mengchu:
I love you.

Acknowledgments

I would like to thank Bing Li. I was lack of confidence when I was a master student. It was him who encouraged and provided me research opportunities. I learnt most of my research abilities from him. For an inexperienced researcher as I was, doing research was like falling into a dark forest surrounding by fog – everything was so uncertain and there was no clue to find an exit. I am extremely lucky to have him accompany me, guide me, and train me not to fear the darkness.

I would like to thank Prof. Tsung-Yi Ho. He brought me to microfluidic design automation field. He knows the value of my work, and always help me to promote my work.

I would like to thank Prof. Ulf Schlichtmann. He is my doctoral advisor, and the best advisor that I can ever imagine. He gives me his all trust, so that I can concentrate on my research. The things I've learnt from him are more than technical knowledge: I've also learnt the way to write, to speak, and even to think. I wish I can become a person like him.

I would like to thank Mengchu Li. She is always with me, and gives me advices from life to research. Our every moment, even arguing about different research opinions, is my treasure – I am so lucky to find Mengchu, from so many people in the world.

Prologue

As I was an undergraduate student, I worked on the multi-threshold complementary metal–oxide–semiconductor (MTCMOS) power switch routing problem[38], which was a typical routing problem for digital circuits with a point-to-point distance limitation. After graduation, I went to Germany to pursue my master degree in Technical University of Munich (TUM), where I worked on a printed-circuit-board (PCB) routing problem[39,40], which was a single-layered routing problem with length-matching constraints.

I then became a doctoral candidate in December 2013. In addition to my main research topic – design automation for continuous-flow microfluidics, I keep doing research on design automation for electronic circuits: I have improved my method for the PCB routing problem[41], and started to work on layout generation for radio frequency integrated circuits (RFICs)[45,46]. The research experience from working on various topics significantly broadens my view, and has become a corner stone for my research in design automation for continuous-flow microfluidics.

1. Introduction

One of the most fundamental demands of humans is to live a long life, and one of the biggest obstruction of living a long life is disease. Diseases can be caused by external and internal factors. With the development of modern medicine, people have learnt more and more about external factors such as viruses and bacteria, and come up with many effective methodologies to deal with them. But many dangerous diseases, including cancer, are caused by internal factors that we are not very familiar with: genes.

A gene is made up of nucleotides and is a protein coding sequence of DNA. The human genome consists of tens of thousands of genes, encoding the genetic information, which guides our body to make all the needed proteins that enable us to live and grow. By understanding genes, people can understand how our body works, and figure out the factors that prevent our body from working properly.

In order to decode the genetic information, many researchers have devoted themselves to genetic analysis such as DNA sequencing, polymerase chain reaction (PCR), reverse transcription polymerase chain reaction (RT-PCR), etc. For example, in 1990, the U.S. government launched the Human Genome Project (HGP), which is the largest undertaking in the history of biological science[36]. The goal of this project was to determine the sequence of nucleotides in our genome, and to

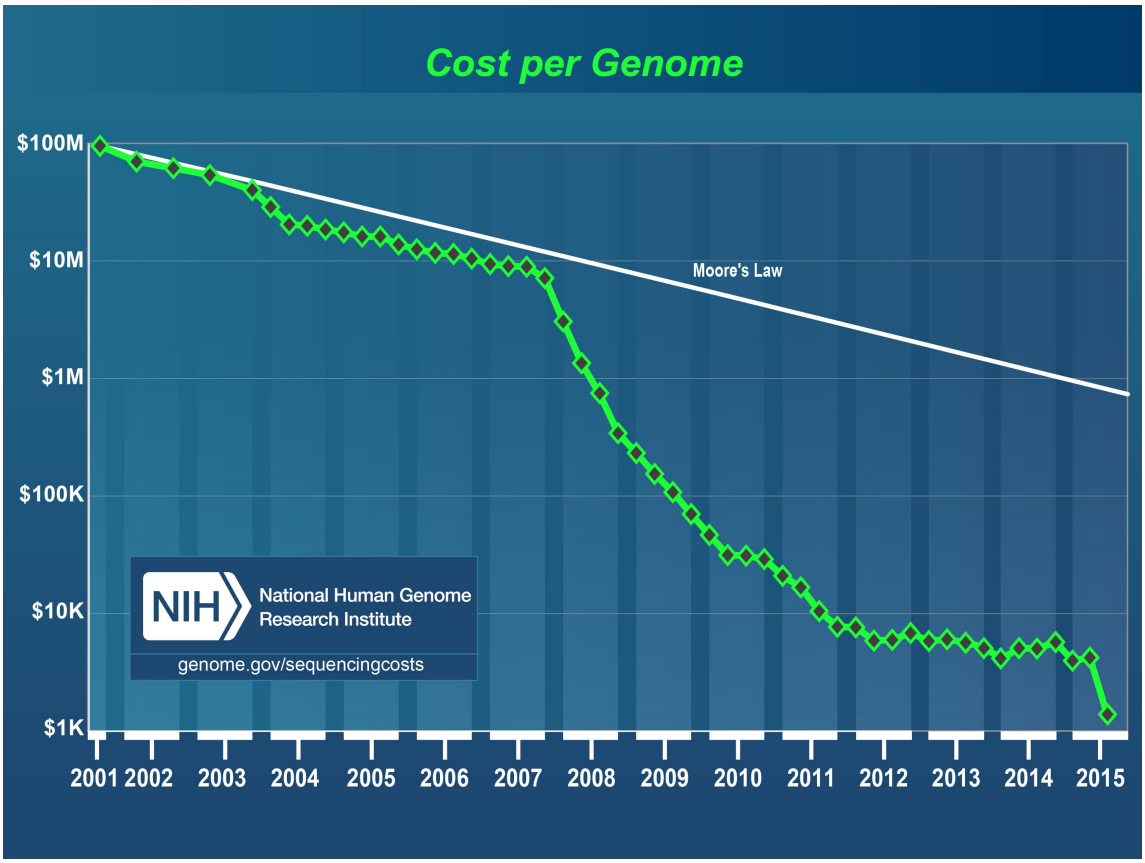

Figure 1.1: Cost change per human genome.[26]

identify their locations as well as their functions. HGP was declared complete in 2003, coming up with the sequence of the majority of the human genome. This project spent 13 years along with 3.8 billion dollars[27], since most of the technologies then were low throughput and extremely cost-prohibitive. As shown in Figure 1.1, in 2001, the 11_{th} year of HGP, it cost about 100 million dollars to sequence the genome from one single person. This cost decreased over time: from 2001 to 2007, the speed of cost reduction basically followed the Moore's Law. Then a sudden drop of the cost began in early 2008: the cost decreased with a dramatic speed and after seven years, in 2015, it cost only less than 2000 dollars

to sequence one human genome.

This sudden drop owes to a technology evolution, according to the U.S. National Human Genome Research Institute (NHGRI)[26]. Before 2008, DNA sequence was generated using Shotgun sequencing method and capillary-based instruments, which was also referred to as the first-generation sequencing technology. Beginning from January 2008, the second-generation sequencing technology has been applied to generate DNA sequence. The second-generation sequencing technology includes a variety of methods developed by different groups and companies, e.g. Illumina dye sequencing, by Illumina, Inc, ABI SOLiD sequencing, by Life Technologies, 454 sequencing, by 454 Life Sciences, etc. One thing these different methods have in common is that they all are based on the same platform: microfluidic biochips.

A microfluidic platform or microfluidic biochip comprises an easily combinable set of microfluidic devices, which can be treated as mini-platforms for basic biological operations such as fluid transportation, fluid metering, fluid mixing, etc[21]. Instead of discrete cumbersome laboratory instruments, these microfluidic devices are connected with each other or even integrated monolithically to a single chip, and can be automatically controlled by customized software. Thus, microfluidic platforms enable the miniaturization, integration, automation, and parallelization of biological assays, and have advantages over conventional laboratory methods in high throughput processing, cost-saving, ease of controlling and reliability.

Besides DNA sequencing, microfluidic biochips are also widely applied to other gene-analysis assays including *single-cell isolation*[19,50], *chromatin immunoprecipitation*[51,52], *polymerase chain reaction (PCR)*[4,13], etc. Among different branches of microfluidic technologies, continuous-flow microfluidic biochip is the

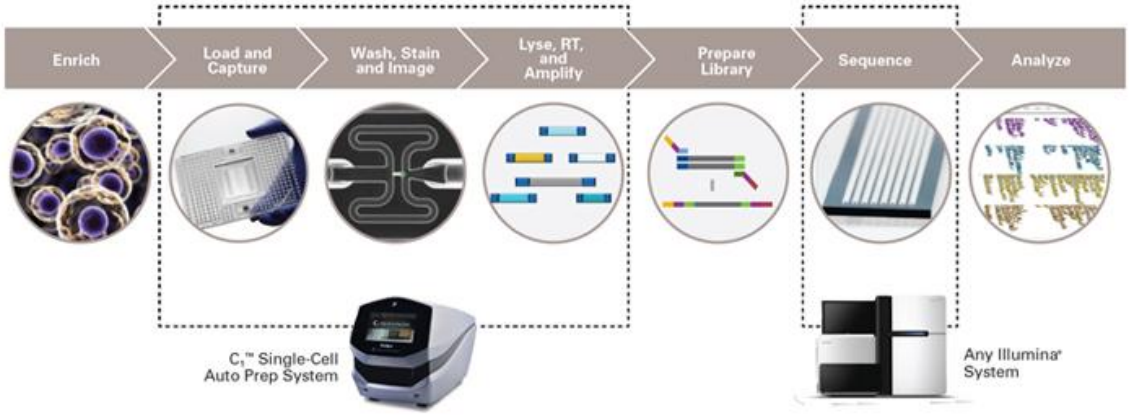

Figure 1.2: A typical workflow for current single-cell cDNA synthesis and sequencing.[28]

mainstream approach due to its ability of precise control[5]. Figure 1.2 shows a typical workflow for current single-cell cDNA synthesis[28], in which various basic biological operations including cell loading, cell capturing, washing, staining, imaging, cell lysis, reverse transcription (RT), and amplifying are executed in one single microfluidic platform: the Fluidigm's C1 Single-Cell Auto Preparation System[8], which applies continuous-flow microfluidic technology.

Most continuous-flow microfluidics are application specific: the manipulation of bioassay protocols usually requires a new design of the chip. Since continuous-flow microfluidics have a multiple-layered structure with complex interactions among microfluidic devices, the design task becomes a heavy burden. The current mainstream design method is to draw the chip layout manually by drawing tools such as AutoCAD. However, manual design is error-prone, time-consuming, and often does not yield optimal results. Designers need to be extremely careful and to be experts in both biology and engineering. It is common for a specialist to spend several weeks or months to design a single chip with fewer than 100 microfluidic devices. The heavy design burden also becomes an obsta-

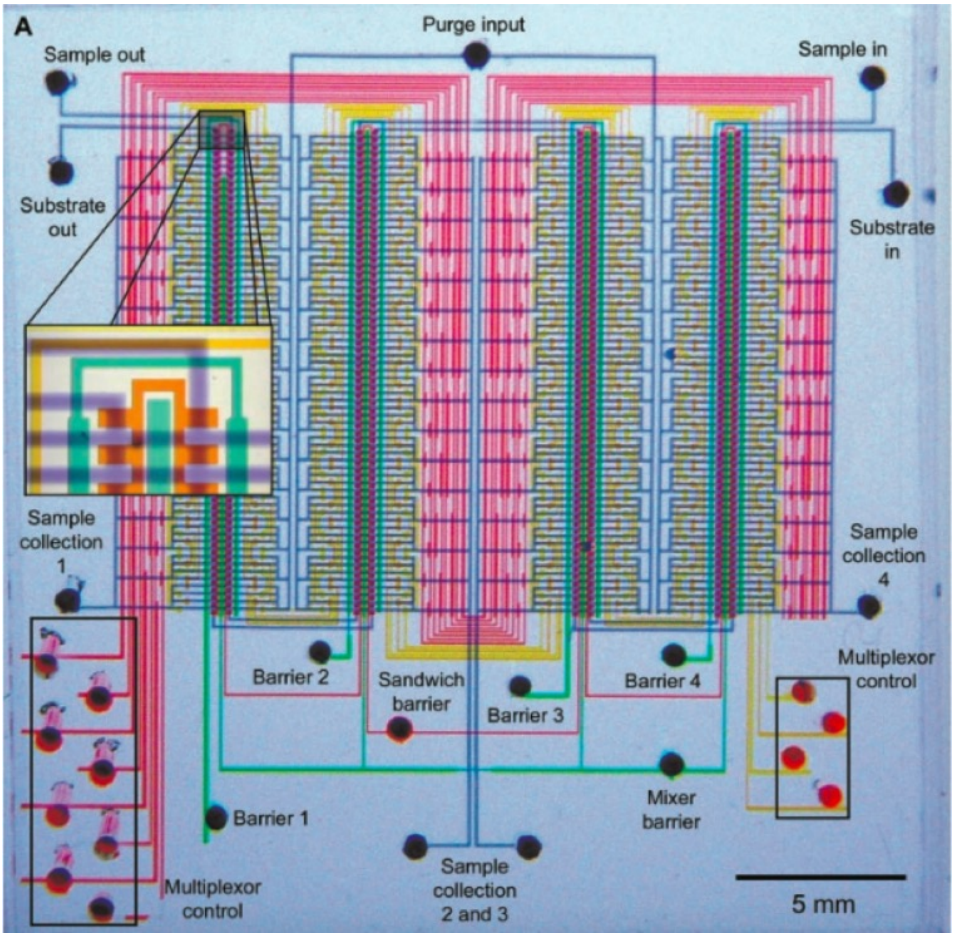

Figure 1.3: Optical micrograph of the microfluidic chip integrated with thousands of valves and 256 chambers.[35]

cle to large-scale integration. The first large-scale integration on continuous-flow microfluidics was proposed in 2002[35], with thousands of micromechanical valves and hundreds of individual addressable chambers. As shown in Figure 1.3 , the proposed chip had a homogeneous structure to alleviate the design effort, which however limited the functionality of the chip. In the last decade, despite the rapid evolution of continuous-flow microfluidic technology, large-scale integration has remained a theoretical target.

Also in the last decade, several design automation engineers, who used to focus on the design automation for electronic chips, began to pay their attention to the design automation for microfluidic chips. With their experience in large-scale integration and chip design, design automation engineers have proposed a series of automatic synthesis methods for microfluidic designs, aiming to solve many practical concerns including the scheduling of basic biological operations[23][24], mapping operations to microfluidic devices[14], chip layout generation[47], reliability[42] and testing[12] of the chip. The systematic research methods inherited from electronic engineering enable this new field to grow fast, and people are getting close to a design automation solution to the microfluidic bottleneck.

As research continues, the main challenge in design automation for microfluidics becomes the knowledge gap between biology and engineering. Continuous-flow microfluidic biochips and biological assays have many unique properties, which cannot easily be formulated as classical problems in electronics engineering. Before we can automatically generate a continuous-flow microfluidic design that can be directly applied to bioassay execution, there remain a number of practical concerns to be dealt with.

In addition to demonstrating my achievement in my research field, another main target of this dissertation is to provide a review about the current design automation progress for continuous-flow microfluidic biochips. I will give an introduction to the most influential works proposed by other research groups in this field, and discuss about their contributions as well as their inadequacies in Chapter 3. From Chapter 4 to Chapter 8, I will discuss my research. I aim to work out an automated complete design methodology that covers both front-end and back-end design of continuous-flow microfluidics. I have proposed high-level synthesis solutions including scheduling and operation-device mapping, taking

consideration of realistic concerns such as extended microfluidic component library, intermediate sample caching, and operation execution limitations[14,43]. I have also proposed the first top-down synthesis methodology for reconfigurable matrix-shaped microfluidic structure, by proposing a dynamic device mapping concept[42,44]. For the mainstream continuous-flow microfluidic structure, I have developed the first automatic co-layout synthesis tool named Columba[47], which takes net descriptions as inputs, and synthesizes the layout of both control and flow layers simultaneously. The output of Columba is an AutoCAD-compatible design, which is ready for mask fabrication.

2. Background

2.1 Structure of Continuous-flow Microfluidics

Current design automation research for continuous-flow microfluidics deals with microfluidics based on multi-layered valve technology[48], which is typically fabricated using Polydimethylsiloxane (PDMS) material. In such chips, continuous flows are generated by external or internal pressure sources through microchannels. These channels are distributed to different layers for the execution of different tasks: channels, through which reaction samples and reagents are transported and operated, are called *flow channels*; and channels, through which fluid pressure or gas pressure is transported, are called *control channels*. The layer, where flow channels are fabricated, is called *flow layer*; and the layer, where control channels are fabricated, is called *control layer*.

The precise control of fluid transportation in a chip is realized by valves. A valve consists of channel segments from both flow layer and control layer, which are separated by a membrane. The flow channel segment of a valve has a rounded profile, so that it can be completely blocked by the shape change of the membrane. Figure 2.1(a)(b) shows the structure of a push-up valve, where the flow layer is fabricated over the control layer. We call a valve "open", when no pressure comes through the control channel segment of the valve. In this situation, there is no

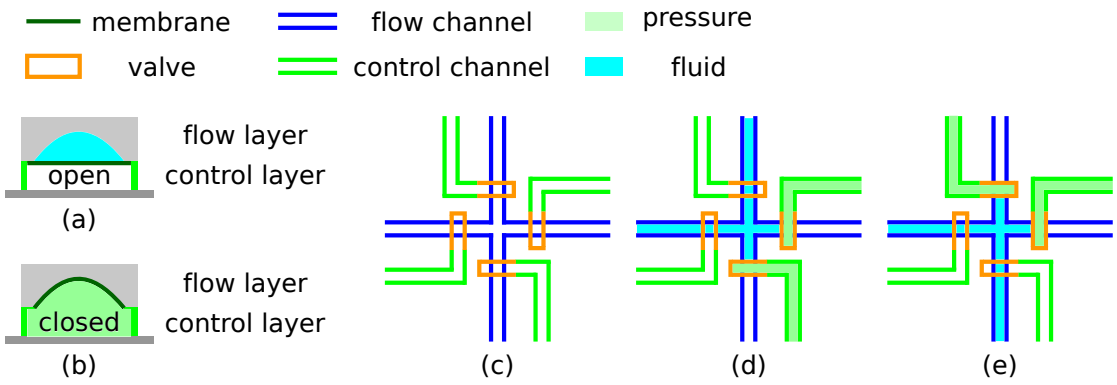

Figure 2.1: Structure of: (a) an open push-up valve. (b) a closed push-up valve. (c) a four-end switch. (d) (e) different flow paths formed by different valve status.

shape change of the membrane, and fluids can pass through the corresponding flow channel segment without obstruction, as shown in Figure 2.1(a). But if sufficient pressure is applied through the control channel segment to the membrane, the membrane will deflect upwards, and thus block the path in the corresponding flow channel segment so that the valve is "closed", as shown in Figure 2.1(b). The actuation of valves determines the fluid direction when two flow channels cross. For example, Figure 2.1(c) shows the structure of a typical microfluidic *switch* that guides fluid transportation. This switch consists of two crossed flow channel segments and four valves, each of which is connected to a control channel. By applying pressure to two of the four valves through different control channels, different flow paths can be formed as shown in Figure 2.1(d)(e).

Flow channels and control channels form not only platforms for fluid transportation, but also platforms for fluid operation. Figure 2.2(a) shows the structure of a rotary mixer, which is the platform for the execution of mixing operations. Mixers are widely applied in many applications such as PCR[16] and cell lysis[51]. In this structure, a flow channel segment is connected end to end to form a ring. On the top side of this ring, there is a group of valves that are connected in series

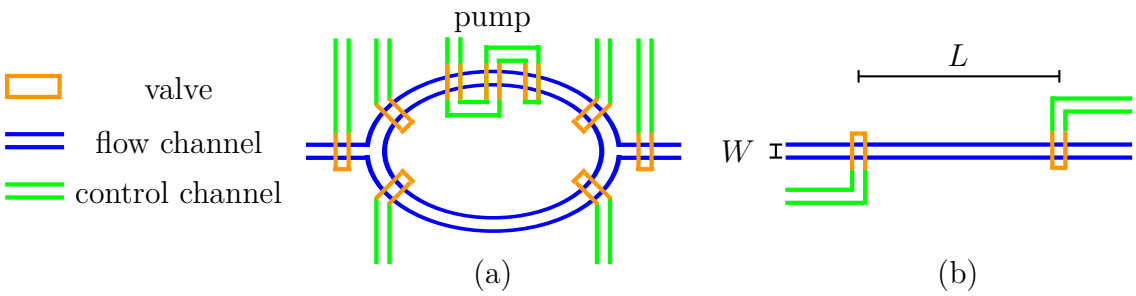

Figure 2.2: Structure of: (a) a rotary mixer. (b) a reaction chamber for volume metering operation.

and share the same pressure source. These valves form a peristalsis pump: when the control channel connected to these valves is inflated, the valves will be closed one by one from left to right, generating a clockwise flow; and then the channel is deflated, which again opens the valves one by one from left to right, sucking the fluids clockwise and thus forming the circulation. As the peristalsis pump enables the rotary motion of reagents and samples inside a ring, the mixing efficiency is significantly enhanced compared with conventional dilution.

Another important microfluidic component for operation execution is a reaction chamber, which can be applied in a variety of applications such as single-cell isolation[20], mixing[50], amplification[49], neutralization[19] and cell culturing[9]. Figure 2.2(b) shows the structure of a reaction chamber for volume metering operation, which is a basic microfluidic operation that separates a specific volume of fluids from a large amount of samples or reagents. This reaction chamber is constructed with one flow channel segment and two valves. By controlling the width W of the flow channel and the distance L between the two valves, fluids of volume $W \times L$ can easily be separated.

In this dissertation, we refer to microfluidic components fabricated for the execution of operations, such as mixers, and reaction chambers as *microfluidic devices*, sometimes *devices* for short.

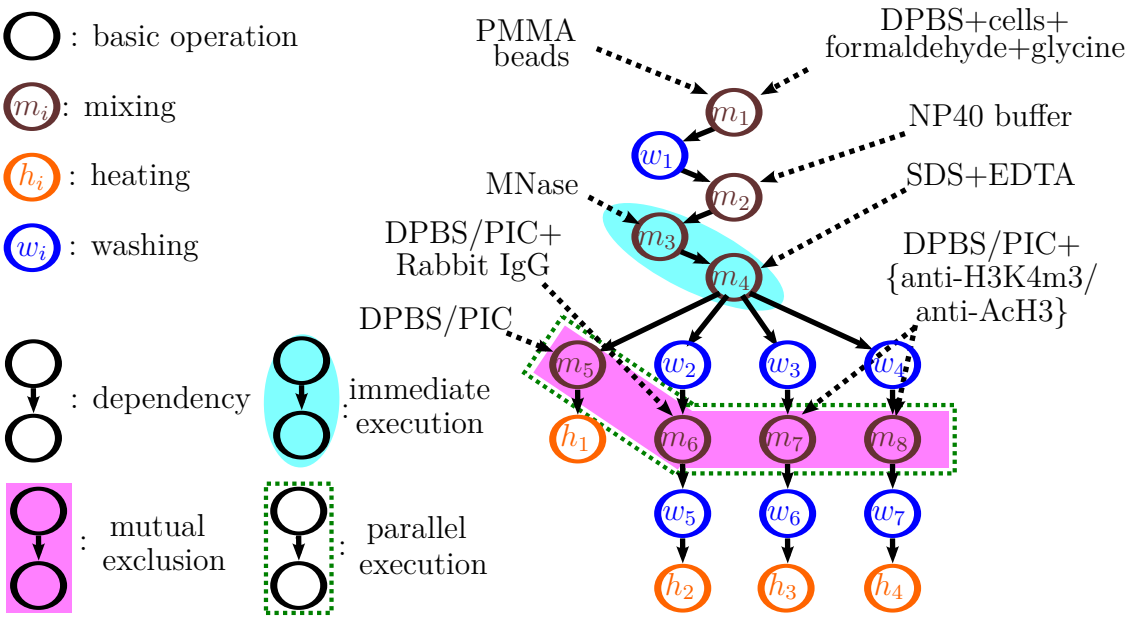

Figure 2.3: Protocol of a chromatin immunoprecipitation (ChIP) application.[14]

Besides channels and valves, a continuous-flow microfluidic biosystem may also consist of other accessory components such as heating pads[16], and external laboratory apparatus such as themocyclers[51] and microscopes[55]. But the core of the design automation technology for continuous-flow microfluidics still focuses on the usage and arrangement of on-chip resources. This can be generalized as two research fields: front-end research and back-end research.

2.2 Research Field: Front-End

Most continuous-flow microfluidics are application-specific, which means that a chip is usually fabricated for a specific biological application. A biological application may consist of several basic biological operations. For example, Figure 2.3 shows the protocol of a chromatin immunoprecipitation (ChIP) application[14], which can be decomposed into sequential operations including mixing,

washing, and heating. In this dissertation, we refer to a biological application that a microfluidic chip is fabricated for as an *assay*, and we refer to the basic biological operations that construct an assay as *operations*.

Front-end design automation research aims to determine the usage of on-chip resources and provide a guidance for assay execution. Major research topics in this field include: interpretation of assay protocols; operation scheduling & microfluidic component mapping; and trade-off optimization.

Interpretation of assay protocols is to decompose an assay to different operations and summarize all the key factors that influence the assay execution. A proper interpretation should give a clear definition for each operation, including its duration, type, dependency relationship with other operations, and specific execution limitations such as immediate execution, with which sequential operations should be executed with little transition time to prevent overreaction, mutual exclusion, with which operations should not be executed in the same microfluidic component to prevent contamination, and parallel execution, with which operations should be executed in parallel to achieve a fair comparison.

Based on the interpreted assay protocols, operations can be scheduled and mapped to corresponding microfluidic components for execution. Specifically, the time spent in performing an operation must match the operation duration given in protocols, and the type of the microfluidic component that an operation is mapped to must match the operation type. Operations that depend on the completion of other operations must be scheduled after the completion of the corresponding operations, and the scheduling and mapping results of an assay must not violate its execution limitations.

Since the execution of an operation requires exclusive occupancy of its corresponding microfluidic component, there is a trade-off between on-chip resources

and execution efficiency. More on-chip resources enables higher assay throughput and thus better execution efficiency, but also increases the design and fabrication difficulty, since the chip area is limited. Therefore, it is essential to optimize the usage of on-chip resources according to the need.

An important issue that must be considered in front-end research is fluid transportation. Compared with the transportation of electronic signals, the transportation of fluids is much more time-consuming. For example, it may take 5 seconds for water to pass through a 100μm long flow channel segment[7], which means that in a 20mm×30mm chip, the transportation time for water between two distant microfluidic components may easily surpass several minutes. Besides, since the volume of fluids cannot be ignored, it also raises practical concerns such as the occupancy of flow channels, and the storage of intermediate operation products.

2.3 Research Field: Back End

Back-end design automation research, or so-called physical design, for continuous-flow microfluidics aims to design the chip layout, including the placement of microfluidic devices, and routing of microchannels.

Microfluidic devices are execution platforms of operations. Common microfluidic devices include mixers and reaction chambers, which have relatively fixed structures as shown in Figure 2.2(a)(b). The exact dimension of devices depends on the volume of samples and reagents needed for the corresponding operations. The placement problem for continuous-flow microfluidics is to determine the dimension, orientation and location of devices, and the routing problem is to route the microchannels between these devices, either based on the placement result or considered together with the placement problem.

As mentioned in Section 2.1, continuous-flow microfluidics consists of control channels and flow channels. Control channels are fabricated in the control layer for pressure transportation, and flow channels are fabricated in the flow layer for fluid transportation. Therefore, the routing problems for continuous-flow microfluidics can be divided into two single-layered sub-problems: control channel routing and flow channel routing.

Control channel routing for continuous-flow microfluidics is to route paths among valves and control inlets. In continuous-flow microfluidics, pressure is transported from external pumps via on-chip inlet ports, each of which occupies about $1mm^2$–$3mm^2$ chip area[30]. Since the area of a chip is limited, the number of inlets on a chip is limited, too. In complex designs, some valves are connected by control channels directly to inlets, and others are connected together by control channels to share the pressure inlets. Control channel routing problems include determining the pressure sharing relationship among valves, and guaranteeing the corresponding pressure paths without undesired overlapping of control channels.

Flow channel routing for continuous-flow microfluidics is to route the incoming and outgoing fluid transportation paths of microdevices for operation execution. In this dissertation, we also refer to the incoming fluids of a device as the *inputs* of the corresponding operations, and the outgoing fluids of a device as the *outputs* of the corresponding operations. In continuous-flow microfluidic assays, operation inputs can either be pre-treated samples and reagents, or reaction products of other on-chip operations. Therefore, the incoming flow channel segments of a device can either be connected to fluid inlet ports, or to the outgoing flow channel segments of other devices. Similarly, the outgoing flow channel segments of a device can either be connected to fluid outlet ports, or to the incoming flow channel segments of other devices. Since the connection relationship among de-

vices and chip ports is usually determined before the routing process, the flow channel routing problem concentrates on finding feasible routing solutions.

As mentioned in Section 2.1, flow channels are allowed to cross. Wherever two flow channels cross, there must be a switch to guide the fluid direction. Since switches consist of valves, the implementation of switches also means the implementation of extra control channels and even pressure inlets to control the valve actuation. In this manner, the flow channel routing problem and control channel routing problem interact with each other, which complicates the whole routing problem.

3. Current Status of the Design Automation Research for Continuous-Flow Microfluidics

In the last decade, design automation research for continuous-flow microfluidics made significant progress. In this chapter, I would like to introduce the development course and status of this field, by introducing the most influential work in major research areas.

3.1 Pioneers: From Digital Circuits to Microfluidics

Though continuous-flow microfluidics is the mainstream approach in current microfluidic technology, design automation research for microfluidics first concentrated on another type of microfluidics: digital microfluidics[32]. Different from continuous-flow microfluidics, which has a heterogeneous structure consisting of various fixed microfluidic devices, digital microfluidics has a homogeneous structure consisting of electrodes, which form *virtual devices* that can be reconfigured to change their functionality during assay execution. Due to its homogeneous structure, digital microfluidics shows great potential of large-scale integration,

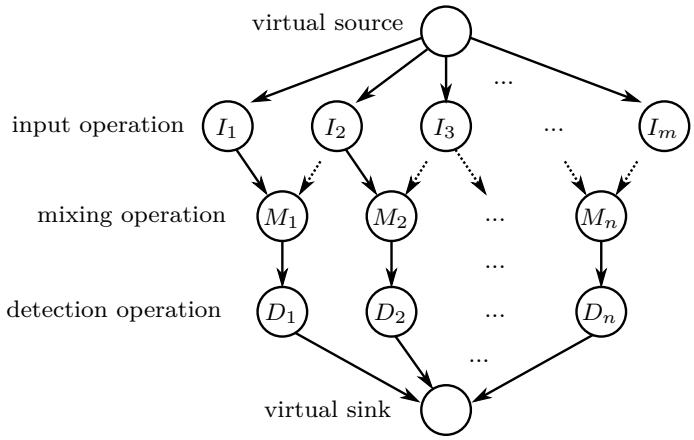

Figure 3.1: Sequencing graph model of a biomedical assay.[31]

and thus attracted much design automation interest. The development of design automation research for digital microfluidics provided valuable experiences that were borrowed by later design automation research for continuous-flow microfluidics.

F. Su's works[31 32 34] from 2004 to 2006 were the most influential works in early design automation field for digital microfluidics. This series of works proposed a system level top-down design automation methodology that included the interpretation of bioassay protocols, operation scheduling, resource mapping, module placement & routing, and testing. It literally covered all the important research topics in design automation for digital microfluidics. Although digital microfluidics has a different layout structure and fabrication technology compared with continuous-flow microfluidics, both types of microfluidic chips perform similar biological applications. Therefore, the front-end research problems in both research fields are similar. Su's work[31] translated the front-end research problem into a typical high-level synthesis problem focusing on scheduling, and applied mature design automation techniques for digital circuits to solve the problem.

Complex bioassay protocols were modeled as sequencing graphs as shown in Figure 3.1. Each node in the sequencing graph represented an operation of a specific type, and each edge that combined the nodes in the graph represented the dependency between two operations. Su classified operations as three types: *input operations*; *mixing operations*; and *detection operations*, and allocated these operations to corresponding time slots in the execution schedule, as well as mapping them to virtual devices re-configured at assay run time.

Although digital microfluidics varies from continuous-flow microfluidics in chip structure and fabrication technology, the sequencing graph model Su proposed to interpret complex assay protocols was widely accepted in the design automation field for continuous-flow microfluidics. A. M. Amin's work[1] in 2007 applied the type classification concept from Su, and modified it to adapt the properties of continuous-flow microfluidics. A. M. Amin classified the continuous-flow microfluidic components into three types: *reservoir* (temporary fluid storage); *fluid functional units* (abbreviated as *FFU*, microfluidic devices); and *routing components* (valves and channels). A. M. Amin also strengthened the bonds between operations and microfluidic devices (FFUs). Different from Su's work, where a new device would be configured for each operation, A. M. Amin classified the FFUs to several types consistent with the operation types, and mapped operations to existing devices of corresponding types. Based on these settings, A. M. Amin's work envisioned a programmable lab-on-a-chip (PLoC) and proposed a fluidic instruction set, called AquaCore Instruction Set (AIS), and a fluidic microarchitecture, called AquaCore, to implement AIS. A. M. Amin assumed that there was a compiler that could automatically translate high-level assay protocols into AIS. The comparison between the protocol of a PCR assay and its hand-compiled AIS code is shown in Table 3.1. Similar to a computer instruction set,

PCR
1. Heat the PCR mixture to 95°C for 5s
2. Heat the mixture to 53°C for 15s
3. Heat the mixture to 72°C for 10s
4. Repeat the thermal cycling 20 times
5. Send the mixture through the capillary electrophoresis (CE) column (5cm at 236V/cm)
6. Separate using separation medium for 180s
7. Sense the fluorescence of the separated flow
	Input port ip1; PCR mixture
	Input port ip2; CE separation medium
	RESULT(); dry array for final results
	PCR{
	input s1, ip1
	input s2, ip2
	move heater1, s1; 5s
	dry-mov r1, 20
	dry-label loop:
	incubate heater1, 95, 5; 6s
	incubate heater1, 53, 15; 17s
	incubate heater1, 72, 10; 12s
	dry-dec r1
	dry-bgt loop
	move separator1.buf, s2; 5s
	move separator1, heater; 5s
	separate.CE separator1, 236, 5, 180
	sense.FL sensor1, RESULT; 180s
	}
	Total time = 895s
	#reservoirs =2
	ASLoC area = unknown (length is 14.5mm)

Table 3.1: Comparison between source assay protocol and AIS (AquaCore Instruction Set) code.[1]

AIS handled one operation per instruction. For example, **mix** *x, t* requires the execution of a mixing operation in a mixer x for t time. Fluid transportation among devices was handled by **move-abs** and **move** instructions, which were specified with start device, end device, transportation time, fluid volume (for move-abs), or relative fluid volume (for move).

The AquaCore Instruction Set did not spread out in the continuous-flow microfluidic research field, since the PLoC that AIS based on had not been invented so far. The PLoC A. M. Amin envisioned in this work was a programmable microfluidic design that could run any assay, which was supposed to be able to

Flow language		Input Point	Output Point	Constraints
ISA	$:= F$	$\text{in}(F)$	$\text{out}(F)$	$\text{in}(F) = source$ and $\text{out}(F) = sink$
F	$:= P_1 \rightarrow P_2$	P_1	P_2	$P_1 != P_2$
	$\mid F_1 \rightarrow F_2$	$\text{in}(F_1)$	$\text{out}(F_2)$	$\text{out}(F_1) = \text{in}(F_2)$
	$\mid F_1 \bigvee F_2$	$\text{in}(F_1)$	$\text{out}(F_1)$	$\text{in}(F_1) = \text{in}(F_2)$ and $\text{out}(F_1) = \text{out}(F_2)$
	$\mid F_1 \bigwedge F_2$	$\text{in}(F_1)$	$\text{out}(F_1)$	$\text{in}(F_1) = \text{in}(F_2)$ and $\text{out}(F_1) = \text{out}(F_2)$
	$\mid F_1 \bigvee \text{mix}(F_2)$	$\text{in}(F_1)$	$\text{out}(F_1)$	$\text{in}(F_2) = \text{out}(F_2)$
	$\mid F_1 \bigwedge \text{mix}(F_2)$	$\text{in}(F_1)$	$\text{out}(F_1)$	$\text{in}(F_2) = \text{out}(F_2)$
	$\mid \text{pump}(F)$	$\text{in}(F)$	$\text{out}(F)$	

Table 3.2: Language for specifying a microfluidic Instruction Set Architecture.[2]

reduce the design effort and enhance the productivity. However, continuous-flow microfluidic technology is undergoing rapid development. New assays are continuously introduced to this field, which require various microfluidic devices. Current fabrication technology is not mature enough for a quasi universal microfluidic platform integrated with most of the needed devices. Though the AquaCore Instruction Set might be premature, A. M. Amin's work gave a very nice introduction of the properties and important mechanisms of continuous-flow microfluidics. The type-classification and type-matching concept proposed by A. M. Amin set up a framework of the design automation solutions for continuous-flow microfluidics, and thus had significant influence on later research.

Besides front-end work, early researchers also proposed design automation solutions to solve back-end problems, or so-called physical design problems. In 2009, N. Amin proposed a language[2] to specify a microfluidic Instruction Set Architecture (ISA). As shown in Table 3.2, the proposed language described the desired flows for executing an assay. Each flow F was specified with a start point *in(F)* and an end point *out(F)*, and an instruction could describe either a simple flow connecting two points ($P_1 \rightarrow P_2$), or a sequential flow connecting two flows in sequence ($F_1 \rightarrow F_2$). The language was proposed to support a control-layer layout generation tool named *Micado*, which was the first automated layout-generation

tool for continuous-flow microfluidics.

The layout generation process of Micado consisted of three phases. The first phase required a drawing of a flow-layer layout and an ISA describing flows, and then Micado could automatically place valves on the control layer to form switches that guide the desired flows. Pressure sharing among valves was also considered in this phase to reduce the number of control channels. The second phase required the designer (user) to indicate the number and locations of control in-/outlets on the intermediate results of the first phase, and then Micado could automatically route the paths among the valves and the control in-/outlets. The last phase of Micado exported a graphical user interface (GUI) for operating a chip at run time.

As the first attempt at automating the layout design of continuous-flow microfluidics, Micado was impressive. N. Amin offered Micado as a free AutoCAD plug-in to bioengineers, and demonstrated its routing ability on three realistic chips. However, Micado was not widely applied among bioengineers to alleviate the design effort, mainly due to two reasons:

First, Micado did not provide a full solution that covered the design of both control and flow layers, but required a fixed flow-layer layout as its input. Thus, the placement of valves had very limited options, and the control-layer design task might suffer from a bad flow-layer design.

Second, the proposed Instruction Set Architecture was at channel level, which meant that the users must specify all the flows for executing an assay, including every intermediate point of a flow path. However, for the sake of scalability, current designs tend to be modularized, and channels forming fixed flow patterns are usually treated as microfluidic devices. Describing detailed channel behavior tends to be a burden to users.

Although Micado did not replace manual labor once for all, N. Amin's work addressed practical concerns of control layer design, and had enlightening significance to continuous-flow microfluidic back-end research.

3.2 Top-down Synthesis for Continuous-Flow Microfluidics

From 2011 to 2013, W. H. Minhass published a series of top-down synthesis works[23,24,25], which modeled the structure of continuous-flow microfluidics and the general characteristics of microfluidic applications in a clear manner.

In this series of works, Minhass proposed an architecture model to allocate microfluidic devices, switches, and fluid paths; and a component model to analyze the operation phase, occupancy, and geometrical dimensions of microfluidic devices. The model for a rotary mixer was informative and especially helpful for later researchers. Based on the architecture model and the component model, Minhass proposed a top-down synthesis method, which started with operation scheduling and operation-device mapping. The result was then analyzed to generate the input netlist for architectural synthesis, which output a schematic design. According to the schematic design, fluid transportation was scheduled and bound to candidate flow paths. Based on the proposed synthesis method, Minhass extracted the valve actuation sequencing for executing an assay, and suggested that valves with the same actuation sequencing could be clustered together to reduce the number of control in-/outlets.

Although Minhass gave a nice introduction of the structure and functionality of continuous-flow microfluidics, the proposed method was not practical enough. In the architecture synthesis, placement and routing were separated into two individual steps. As channel crossings were not considered in the device placement

phase, a large number of switches can be introduced in the final layout. Redundant switches could lead to redundant valves as well as control channels connected to these valves, which were neglected in these works. Moreover, Minhass's work synthesized valve sequencing without the consideration of control layer layout. Arbitrarily clustering valves could lead to heavy burden of control channel routing, since crossing of control channels must be avoided.

3.3 Physical Design for Continuous-Flow Microfluidics

Up to 2015, several approaches had been proposed for flow-channel routing of continuous-flow microfluidics, focusing on different aspects including valve reliability[37], flow paths minimization[15] and length-matching[53] of flow channels. However, the above mentioned works did not take control-layer layout into consideration. Once the technology scaled down and the complexity of a chip increased, the proposed methods could lead to inevitable routing failures in the control layer.

Aware of the drawback of the separated design phase, H. Yao proposed the first flow-control codesign methodology, which performed iterative device placement adjustment to coordinate flow-layer and control-layer design[54].

By taking scheduling and application mapping results as inputs, Yao divided the design flow into 5 phases: 1. initial device placement, which applied a classic simulated annealing method. 2. flow layer routing, which applied a classic A* searching algorithm. The objective of this phase was to minimize flow-channel crossing, and thus to reduce the number of valves needed for implementing switches. 3. device placement adjustment, which adjusted the placement solutions from the previous phases. Devices would be pushed away from a congestion window that contained the largest number of channel crossings, and the

new placement result would be fed back to the last phase. After several iterations, when the number of channel crossings was supposed to be acceptable, the algorithm would move to the control-layer design phases: 4. microvalves addressing, and 5. control-layer routing. In the control-layer design phases, each valve was connected to an individual port, and the escape routing for valves applied the classic A* searching algorithm. When inevitable control channel crossing (routing failure) happened in this phase, a congestion window that contained the largest number of failed valves would be found, and the result would be fed back to the device placement adjustment phase.

As the first work addressing the concern of control-flow layer interaction, Yao's work proposed a practical design flow with solid placement and routing methodologies. However, Yao neglected the area cost of control inlets, and did not consider pressure sharing among valves in the control-layer synthesis. The proposed valve addressing algorithm connected every valve to an individual pressure inlet, which was unrealistic for large-scale designs.

3.4 Testing for Continuous-Flow Microfluidics

As microfluidic technology develops, concern emerges about the lack of testing techniques to detect defective chips. Defective chips result in erroneous operations, which may lead to failure of the whole assay. The standard testing approach for continuous-flow microfluidics is based on visual inspection under microscopes, which suffers from limited effectivity and low efficiency.

In 2014, K. Hu proposed the first approach for automated testing of continuous-flow microfluidics[12]. In this work, Hu focused on two major types of chip defects: *block* and *leak*, which represented disconnection of microchannels, and unexpected

connection of microchannels, respectively. Control and flow channels on microfluidic chips were modeled as logic circuits composed of Boolean gates, and the test generation was carried out using standard ATPG tools. Hu abstracted the chip defects as faulty behaviors of valves, and inferred the condition of valves from external pressure sensors by measuring the pressure in microchannels. By comparing the actual valve condition with the expected valve condition based on the logic circuit model, the types and positions of defects could be identified.

The main difficulty for implementing Hu's testing methodology was the design testability. The design-for-test concept, which is already widely adopted in the electronic chip designs, has not yet been recognized by the designers of continuous-flow microfluidics. One of the main concerns about adopting the design-for-test concept is the number of in-/outlets, which is typically strongly limited as a design rule owing to the characteristics of polydimethylsiloxane (PDMS)[30]. To achieve the testability of valves, Hu's design involved a relatively large number of control outlets, each of which was connected to external pressure sensors. When designs scale up, the proposed methodology may result in increased design difficulty.

4. Temporary Fluid Storage: Flow channel

Microfluidic assays usually consist of sequential operations. Sometimes, sequential operations cannot directly be executed after one another due to the limited number of on-chip devices. Therefore, intermediate products from earlier operations need to be stored. A common approach is to build a dedicated on-chip storage unit consisting of several storage cells, which requires exclusive chip area and additional control efforts. Moreover, a traditional storage unit has only one input and one channel for fluid transportation, which severely hinders the execution efficiency.

I proposed a new approach for temporary storage of intermediate products in 2015[43]. In addition to the dedicated storage unit, I also considered flow channels as temporary caching cells. The concept is illustrated in Figure 4.1. As shown in Figure 4.1(a), o_3 takes the operation products of o_1 and o_2 as its inputs. As shown in Figure 4.1(b)(c), suppose that o_1 is executed in a device d_1, and o_2 is executed in another device d_2. If o_2 completes earlier than o_1, the operation products of o_2 need to be stored. Instead of transporting the products to a dedicated storage unit, we transport them to flow channel c_1, so that d_2 is free to be mapped by operation o_4 for execution. Thus, the execution time of the whole

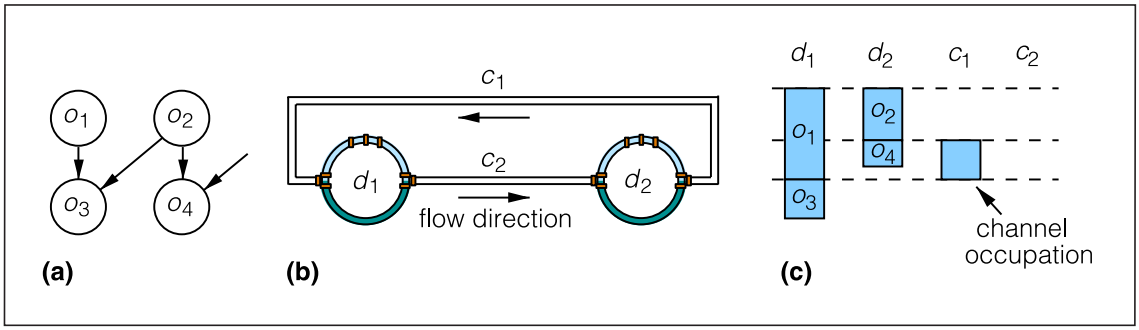

Figure 4.1: (a) operation dependency. (b) microfluidic structure consisting of two mixers and two flow channel segments. (c) scheduling and application mapping results.[43]

assay is shortened by saving the transportation time from and to an extra storage unit. Another benefit brought by caching fluids in flow channels is the reduction of the construction cost of a large central storage and the connections from it to other devices. In the proposed method, I used existing flow channels to store as many fluids as possible. If more than one fluid needs to be stored at the same time, a bypass channel will be created as a distributed storage cell at the spot to accommodate the fluids. In the following, we describe our proposed method in detail. Similar descriptions can also be found in our published papers[14,43].

4.1 Mathematical model for channel caching and storage assignment

We implemented our concept by constructing an integer-linear-programming (ILP) model. The application is given as a sequencing graph (O, E), where O is the set of nodes representing the operations, and E is the set of edges representing the dependency of the operations. For o_i, $o_j \in O$, (o_i, o_j) denotes an edge from o_i to o_j, which represents that the output of operation o_i is one of the inputs of operation o_j, and thus o_j can only start after the completion of o_i. The absolute start

time of the operation o_i is denoted by t_i and its execution duration is denoted by u_i. All the operations in the sequencing graph should be executed by a given set of devices D. With the above settings, the constraints of our model are discussed as follows.

1. *Operation binding*

   Each operation in the sequencing graph should be bound to exactly one device. To handle these constraints, we create $N \times M$ 0-1 variables $s_{i,k}$, where $i = 1 \ldots N$ and $k = 1 \ldots M$. N is the number of operations in the application, and M is the maximum number of available devices. If the i_{th} operation o_i is assigned to the k_{th} device d_k, $s_{i,k}$ is set to 1; otherwise it is set to 0. The actual values of these variables are determined by the optimization solver for an optimal scheduling and assignment. If an operation o_i must be assigned to a device d_k of a specific type, the 0-1 variables $s_{i,k}$ should be set to 0 for all those devices that are not of the required type. With these 0-1 variables, we can write the constraint that the ith operation should be executed only once, or the constraint of uniqueness, as

$$\sum_{k=1}^{M} s_{i,k} = 1, \ \forall o_i \in O. \tag{4.1}$$

2. *Operation duration*

   The operation o_i should have enough time to finish its execution, so that its end time e_i should be at least u_i time later than its start time t_i, where u_i is the duration of o_i. Therefore we have

$$t_i + u_i \leq e_i, \ \forall o_i \in O. \tag{4.2}$$

3. *Operation dependency*

   In the sequencing graph, an edge denoted by (o_i, o_j) means that o_j takes the output of o_i as its input. In this situation, we define o_i as the parent

operation of o_j, and o_j as the child operation of o_i, correspondingly. Since an operation can only start with all its inputs ready, a child operation will not start until the completion of all its parent operations. Assume that the propagation delay from o_i to o_j through the channel is $u_{i,j}$. We can write this dependency constraint as

$$e_i + u_{i,j} \leq t_j, \ \forall (o_i, o_j) \in E. \tag{4.3}$$

This constraint is created for all the edges in the sequencing graph, so that the operations that precede the others on a path are always executed earlier.

4. *Non-interfering operation*

   A new operation should be executed in a free device. If a device is occupied by an operation in progress, other operations should not be bound to this device until the completion of the current operation. Therefore, we introduce the following constraints on every two operations:

$$e_j \leq t_i + \Phi q_{j,i} \tag{4.4}$$

$$e_i \leq t_j + \Phi q_{i,j} \tag{4.5}$$

$$s_{i,k} + s_{j,k} + q_{i,j} + q_{j,i} \leq 3, \ \forall d_k \in D \tag{4.6}$$

   where Φ is a very big constant, $q_{j,i}$ and $q_{i,j}$ are auxiliary 0-1 variables, whose values are determined by the optimization solver in arranging the operations in the devices. Above constraints ensure that operations o_i and o_j can only be bound to the same device, i.e., both $s_{i,k}$ and $s_{j,k}$ are set to 1, when their execution times do not overlap each other (either o_i starts after the completion of o_j, or o_i ends before the execution of o_j).

5. *Channel conflict*

   The condition of channel conflict is illustrated in Figure 4.2, where o_{i_1} and o_{i_2} are assigned to device d_{k_1}, o_{j_1} and o_{j_2} are assigned to d_{k_2}, and these two devices are connected by a channel. In the case on the left hand, the result of o_{i_2} may contaminate the result of o_{i_1} because the latter has not

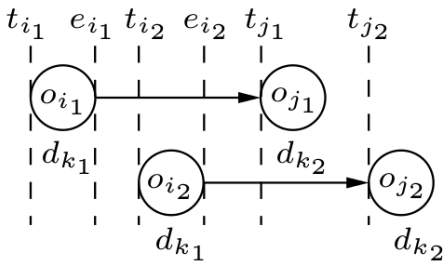

Figure 4.2: Channel conflict scenarios.[43]

entered the device d_{k_2} and still occupies the channel. Similarly, the case on the right shows the mirrored case that o_{i_2} is executed earlier than o_{i_1}. To avoid channel conflicts, either the operation o_{i_2} finishes later than o_{j_1} starts, or o_{i_1} finishes later than o_{j_2} starts. If these conditions can not be met, the operation pairs (o_{i_1}, o_{j_1}) and (o_{i_2}, o_{j_2}) should not share the same pair of devices. Therefore, we can write the constraints to avoid channel conflicts as

$$s_{i_1,k_1} + s_{i_2,k_1} \leq 1 \text{ or } s_{j_1,k_2} + s_{j_2,k_2} \leq 1 \quad \text{if } e_{i_2} < t_{j_1} \text{ and } e_{i_1} < t_{j_2} \tag{4.7}$$

where the condition after if defines the scenarios of channel conflicts shown in Figure 4.2. The constraint $s_{i_1,k_1}+s_{i_2,k_1} \leq 1$ or $s_{j_1,k_2}+s_{j_2,k_2} \leq 1$ guarantees that at least one of the devices for operation pairs is different. (4.7) can be further transformed into

$$(e_{i_2} \geq t_{j_1} \text{ or } e_{i_1} \geq t_{j_2}) \text{ or } (s_{i_1,k_1} + s_{i_2,k_1} \leq 1 \text{ or } s_{j_1,k_2} + s_{j_2,k_2} \leq 1) \tag{4.8}$$

and thus

$$\forall(o_{i_1}, o_{j_1}) \in E,\ (o_{i_2}, o_{j_2}) \in E, \tag{4.9}$$

$$t_{j_1} \leq e_{i_2} + \Phi q_{j_1,i_2} \tag{4.10}$$

$$t_{j_2} \leq e_{i_1} + \Phi q_{j_2,i_1} \tag{4.11}$$

$$s_{i_1,k_1} + s_{i_2,k_1} \leq 1 + q_{i_1,i_2}, \quad \forall d_{k_1} \in D \tag{4.12}$$

$$s_{j_1,k_2} + s_{j_2,k_2} \leq 1 + q_{j_1,j_2}, \quad \forall d_{k_2} \in D \tag{4.13}$$

$$q_{j_1,i_2} + q_{j_2,i_1} + q_{i_1,i_2} + q_{j_1,j_2} = 3. \tag{4.14}$$

The detailed explanation of (4.9)–(4.14) is similar to the case for non-interfering operation above and omitted for simplicity. Note here the complexity of the constraints is still roughly $O(N^2M)$ because the sequencing graph is sparsely connected.

6. *Dedicated storage*

   In the proposed model, dedicated storages are allowed to be contructed as well to store conflicting fluids. We maintain a 0-1 variable sto_k to indicate the existence of the storage for each dedicated device d_k, and relax (4.12) to

$$s_{i_1,k_1} + s_{i_2,k_1} \leq 1 + q_{i_1,i_2} + sto_{k_1},\ \forall d_{k_1} \in D. \tag{4.15}$$

   If the solver determines to insert a storage unit at the output of d_{k_1}, sto_{k_1} is set to 1 so that the constraint (4.15) is always met. The total number of dedicated storages can be expressed as

$$\sum_{d_k \in D} sto_k \leq n_s \tag{4.16}$$

   where n_s is the maximum number of available storage units.

7. *Channel number*

   Assume that operations o_i and o_j have an edge in the sequencing graph, meaning that the result of o_i should be transported to o_j. If these two oper-

ations are assigned to devices d_{k_1} and d_{k_2}, there should be a channel between d_{k_1} and d_{k_2}. A channel can be removed to save chip area or manufacturing effort if it is never used. We maintain a 0-1 variable b_{k_1,k_2} to represent the appearance of the channel between devices d_{k_1} and d_{k_2}. This channel must exist if there are two sequential operations o_i and o_j (with an edge in the sequencing graph) bound to d_{k_1} and d_{k_2} respectively. The channel existence can be modeled as

$$\forall (o_i, o_j) \in E,\, d_{k_1} \in D,\; d_{k_2} \in D, \tag{4.17}$$

$$s_{i,k_1} + s_{j,k_2} \leq 2b_{k_1,k_2} + q_{k_1,k_2} \tag{4.18}$$

where q_{k_1,k_2} is an auxiliary variable. If both s_{i,k_1} and s_{j,k_2} are 1 for at least one single pair of sequential operations o_i and o_j, b_{k_1,k_2} must be set to 1. If $s_{i,k_1} = 0$ or $s_{j,k_2} = 0$, the solver has the flexibility to set b_{k_1,k_2} to 1 or q_{k_1,k_2} to 1. However, since we minimize the number of channels to reduce the cost, in such case b_{k_1,k_2} is always chosen to be 0. With b_{k_1,k_2}, the total number of channels can be described as

$$\sum_{d_{k_1} \in D, d_{k_2} \in D} b_{k_1,k_2} \leq n_c \tag{4.19}$$

where n_c is given as the upper bound of the number of channels.

8. *Objective*

   The aim of scheduling and binding of an application is to find a mapping of operations into given devices, subject to all the constraints described above. The objective of the model is to minimize the number n_s of dedicated storages, the number n_c of channels, and the execution time T of the application, while T is constrained by the end times of all operations as

$$\forall o_i \in O, \qquad e_i \leq T \tag{4.20}$$

   Thereafter, the complete ILP model can be summarized as follows.

Minimize $\omega_t T + \omega_c n_c + \omega_s n_s$
Subject to:
Operation binding (4.1);
Operation duration (4.2);
Operation dependency (4.3);
Non-interfering operation (4.4)–(4.6);
Channel conflict (4.7)–(4.14);
Dedicated storage (4.15), (4.16);
Channel number (4.17)–(4.19);
Maximum application execution time (4.20),

where ω_t, ω_c, and ω_s are tunable coefficients that can be changed according to different optimization emphases.

4.2 Model reduction

Here we discuss two directions of techniques to reduce the computational runtime of solving the ILP model. The first direction is to remove all channel conflict constraints created from edges that are on a path in the sequencing graph, because these edges should be arranged along the timeline as constrained by the precedence condition (4.3) so that channel conflicts never happen between them. The effectiveness of this reduction, however, depends on the structure of the sequencing graph.

The second direction of reducing computational runtime is to provide a general guidance to the solver, so that a solution with good approximation can be reached quickly. In the proposed method, we adapt the List algorithm[22] to find an initial schedule and binding. Thereafter, we use the ILP solver to refine the initial solution gradually to find a better solution, by allowing the solver to adjust

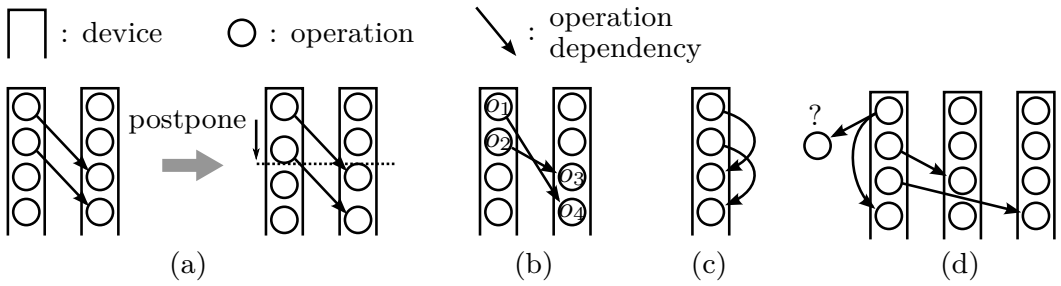

Figure 4.3: Potential conflict scenarios. (a) Resolvable conflict. (b) Crossing conflict (different devices). (c) Crossing conflict (same device). (d) Impending inevitable crossing conflict.

the device binding of operations in a limited range, and to switch the operations along the timeline.

The structure of the adapted List algorithm is shown in the *List* () procedure in Algorithm 1. The List algorithm assigns the next operation to the dedicated device which has the shortest length greedily. When assigning an operation to one of the devices, we should observe the precedence rule to insert child nodes in the sequencing graph after their parents are scheduled. To meet this requirement, we mark the nodes whose parents have been scheduled as active in L3. In the following loop L4–L15, the algorithm schedules these candidate operations, while trying to reduce the chance of channel conflict.

In the greedy scheduling, different types of channel conflict might appear as shown in Figure 4.3(a)–(d). In Figure 4.3, devices are represented by queues and operations are represented by small circles. The execution priorities of operations are denoted by their priorities in the queues. If two pairs of operations are arranged as shown in Figure 4.3(a), a channel conflict would occur but is able to be resolved by postponing the executions of the last two operations in the first device. Another type of channel conflict is shown in Figure 4.3(b), in which the edges between the two pairs of operations cross each other. In this case channel conflict cannot be

Algorithm 1: Adapted List algorithm and layer-based ILP refinement.

```
L1  Proc List ()
L2      repeat
L3          mark_active_operations ();
L4          while there is an active operation do
L5              o_i=random_select_active_operation ();
L6              d_k=check_queue_rules (o_i);
L7              if d_k is valid then
L8                  remove_operation_from_graph (o_i);
L9                  append_operation (q_k, o_i);
L10                 schedule (q_1, q_2,..., q_M);
L11             else
L12                 deactivate_operation (o_i);
L13                 continue;
L14             end
L15         end
L16         if no operation removed from graph then
L17             restore_all_operations ();
L18         end
L19     until sequencing graph is empty;
L20 end
L21 Proc Layer_solve ()
L22     for i ← 1 to n_s do
L23         for j ← 1 to n_l do
L24             solve_ILP (L_{i,j});
L25             if execution time is not improved then
L26                 exit;
L27             end
L28         end
L29     end
L30 end
```

alleviated by simply moving operations along the timeline, and therefore should be avoided in the List algorithm by applying a rule: an operation can only be bound to a device if all the children of the prior operations of its parents are bound to devices already. For example, in Figure 4.3(b), after applying the new rule, operation o_3 cannot be bound to the device before o_4, since only until all the children (o_4 in this case) of the prior operations (o_1) of the parents (o_2) of o_3 are bound to devices already, o_3 is then ready for binding. This *crossing* conflict may occur not only in different devices but also in the same device, as shown in Figure 4.3(c). Similar to Figure 4.3(b), crossing conflict happening in the same device needs to be avoided. Figure 4.3(d) shows an impending inevitable crossing conflict. In Figure 4.3(d), the unqueued operation with a question mark is the child of the first operation in the first device. It is not possible to assign it to any device without causing crossing conflict, and thus we should avoid this situation in advance as well.

In addition to avoiding the above-mentioned crossing conflicts, we also check whether the parents of an operation (or the children of an operation) are assigned to the same devices, which would also result in unresolvable conflict. All these situations are checked by the function check_queue_rules (o_i) in L6. Since the List algorithm greedily inserts operations one by one, if the newly-inserted operation violates the rules, it will be unqueued and deactivated until the next iteration. During this process, if there is no operation eligible to be appended to a queue, a deadlock happens. In this case, all queues are emptied and the current scheduling is discarded. Thereafter, the List algorithm runs again with a new random order of operation selection until all operations are scheduled.

We apply the List algorithm to produce a valid initial scheduling-and-binding result. We then partition this result into several layers along the timeline

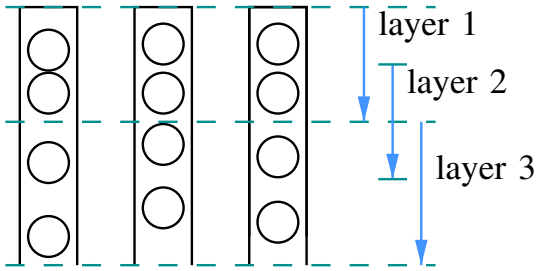

Figure 4.4: Refining the greedy schedule using ILP model across layers.

and optimize each layer by applying the mentioned ILP model, as illustrated in Figure 4.4. Starting from the first layer downwards, the ILP model is applied to adjust the operations by 1) moving them between devices; 2) switching the order of them along the timeline; 3) reducing the time for execution postponement.

To find a good approximation of the optimal solution, we allow the solver to adjust the operations across layer boundaries by layer overlapping as shown in Figure 4.4. During the iterations of optimization, the operations are pushed to earlier execution times gradually so that the execution time of the application can be reduced. This iterative optimization with layered schedules is summarized in Algorithm 1 as the function *Layer_solve* (), where n_l is the number of layers in a partition; n_s is the number of iterations. In each iteration, the layer $L_{i,j}$ is adjusted according to the current schedule and processed using the ILP model as shown in L24. The optimization process finishes after a given number of iterations or no further improvement is made, and the currently achieved scheduling and binding is returned as an approximation of the optimal solution.

4.3 Storage assignment

After the model is solved, the schedule and binding of the biochip is determined. Thereafter, flow transportations should be assigned into physical cells in storage

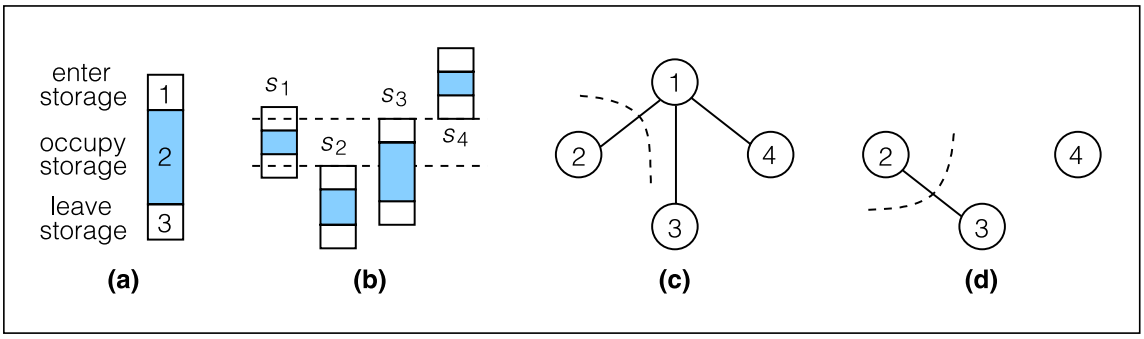

Figure 4.5: Storage assignment. (a) Fluid phases. (b) Fluid conflicts. (c) Port conflict graph. (d) Storage cell conflict graph.[43]

units. If two fluid samples do not have conflicts, they can share the same storage cell by time multiplexing, just as different data can be stored in the same memory cell at different time frames in a digital circuit.

The time which a fluid sample spends in a storage unit can be partitioned into three phases. In phase one, it enters the storage unit, so that the duration of this phase is the time from the device to the port of the storage unit. Since a storage unit usually only allows one fluid sample to enter or leave due to the flow path, only one fluid sample is allowed to be in phase one at a time. Otherwise, a port conflict occurs. In phase two, the fluid sample occupies a cell in the storage unit. In phase three, it leaves the storage unit, and again only one fluid sample is allowed to use the port of the storage. Figure 4.5(a) shows the three phases of a fluid sample occupying the dedicated storage unit, and Figure 4.5(b) shows an example of four fluid samples directed to the dedicated storage unit but with port conflicts.

Since two fluid samples can not enter the same storage unit if there is a conflict at phase one or phase three, we need to find the largest set of fluid samples that do not conflict with each other and pack them into a dedicated storage unit.

The conflict relation between fluid samples can be represented using a conflict graph as illustrated in Figure 4.5(c). In this graph, a node represents a fluid sample. If there is a port conflict between phase one or phase three of any pair of fluids, an edge is created between the nodes. Thus the problem is transformed as to find the maximum independent set of nodes between which there is no edge. Since the maximum independent set problem is NP-hard, we apply a greedy approximation algorithm[17]. In this algorithm, the node with the smallest degree is selected and removed together with all other nodes connected to it. This process is repeated until all nodes are removed from the graph, and the selected nodes together form an independent set. For example, if we remove node 2 together with node 1 in Figure 4.5(c), we can find the independent set containing node 2, node 3, and node 4, the fluids representing by which will then be saved in the dedicated storage unit.

From the independent sample set in the previous step, we need to determine how many cells are required in the dedicated storage unit. As discussed earlier, if two fluid samples do not occupy storage at the same time, they can reuse the same cell in the style of time multiplexing. Similar to handling port conflicts, we try to find the largest set of fluid samples that can share a storage cell using a cell conflict graph, as shown in Figure 4.5(d). In this graph, an edge means that the two fluids have some overlap in phase two, so that they should use different storage cells. The maximum independent set problem is solved using the greedy algorithm[17] again. The result shows that the fluid representing by node 2 can share a cell with the fluid representing by node 4, so that in total two cells need to be built in the dedicated storage unit.

After determining the independent sets in the conflict graphs, we might have fluid samples that cannot be saved into a dedicated storage unit due to port

Assay	List alg. with storage			Storage and caching			
	$\#ch_L$	$\#sto_L$	T_L	$\#ch_p$	$\#sto_p$	T_p	r_p (s)
PCR7	1	2	49.73	1	1	49.73	0.16
MT18	4	1	62.4	4	0	60.84	3.22
PI39	6	7	129.94	5	5	98.57	17.14
PE55	6	10	161.53	6	9	121.38	72.79
gen15	4	4	41.64	3	1	39.73	15.73
gen31	4	6	79.83	4	6	77.59	72.57
gen63	16	19	83.39	14	11	81.62	440.35
gen127	25	32	139.58	23	23	139.18	894.31
average reduction				7.8%	41.7%	7.7%	

Table 4.1: Results with storage and channel caching.[43]

conflict. To solve this problem, we either generate additional dedicated storage units, or create distributed storage cells along channels directly. In the proposed method, we use the latter method, because the number of such fluids is not large and the distributed cells have short transportation time without the complex controlling mechanism at the ports of dedicated storage units. This method can also reduce routing challenges because fewer channels are needed to reach dedicated storage units. The process of generating distributed storage cells is similar to the last step in storage assignment above. We only check which fluid samples can share a distributed cell using the maximum independent set algorithm so that the efficiency of distributed storage cells along transportation channels is maintained.

4.4 Experimental results

The experimental results are shown in Table 4.1. The proposed method was implemented using and tested on a computer with a 2.67 GHz CPU. Four real biochemical cases[33] and four synthetic cases, gen15–gen127, were used for exper-

iments. The List algorithm[31], which does not consider constraints from channels and storage, was implemented for comparison. This algorithm produced schedules for the assays, to which the same maximum independent set algorithm[17] was applied to generate distributed and dedicated storage units.

The columns T_L and T_P are the assay execution times calculated by the List algorithm and the proposed method. From this comparison, we can see that the proposed method resulted in improvements in almost all assays, by 7.7% on average. Specially for PI39 and PE55, the improvement on execution time can reach nearly 25%.

The results of transportation channels and storage cells from the List algorithm are shown in columns $\#ch_L$ and $\#sto_L$, respectively. The results from the proposed method are shown in the columns $\#ch_p$ and $\#sto_p$, respectively. The proposed method does not require more channels or storage cells to achieve the shortened execution time. In cases such as gen63, channels and storage cells are reduced significantly. On average, these reductions reach 7.8% and 41.7%, as shown in the last row of Table 4.1.

To demonstrate the effect of channel caching, a baseline method was implemented. In this method, a device cannot start a new operation before its previous output sample is taken by another device to avoid channel conflicts and thus sample contamination. The execution times calculated by the baseline method and the proposed method are illustrated in Figure 4.6. Clearly, the proposed method effectively reduces the execution time of an assay by simply caching fluid samples in transportation channels. The runtimes of solving the proposed ILP model and the storage assignment for the test cases are shown in the column $r_p(s)$ in Table 4.1. For the largest application with 127 operations, the runtime is 894.31 seconds, largely taken by the ILP solver. These computational runtimes

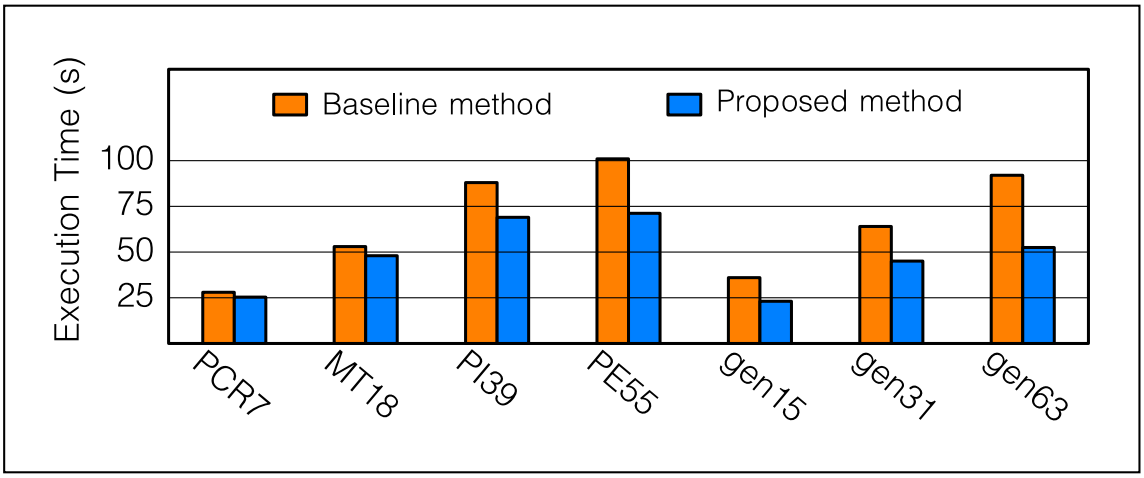

Figure 4.6: Execution times of assays calculated by the baseline method and the proposed method.[43]

are already acceptable for an offline synthesis flow. As a conclusion, the proposed channel caching concept and method can save both total assay execution time and the cost of building a dedicated storage.

In this chapter, a concept to cache fluid samples in transportation channels and synthesize storage cells considering fluid conflicts is explained. By minimizing channel conflicts and recognizing maximum independent sets, storage requirements are handled jointly by channels as well as both distributed and dedicated storage cells. Results show that the execution time of the assay and resource usage are lowered effectively at the same time.

5. Sieve Valve and Execution Limitations

One of the most important front-end design tasks for continuous-flow microfluidics is the interpretation of assay protocols. The current design automation approach abstracts the protocol of an assay as a directed graph. Each node of the graph represents an operation of a specific type, and each edge of the graph represents the dependency between operations. With an accurate type classification strategy, this abstraction simplifies the design process and provides a basis for scheduling and operation-device mapping.

However, in the design automation field, there has been a misreading of an important type of biological operation: washing. Washing operations used to be mistaken for rinsing operations, which are performed to clean microfluidic components to avoid fluid contamination. In fact, washing operations are widely recognized in the biological field as a particular type of operation to extract target particles from suspensions, which is essential in assays involving cells. The misreading of washing operations resulted in the neglect of an important microfluidic component: sieve valve.

The sieve valve is a variety of partially closed valves. As shown in Figure 5.1,

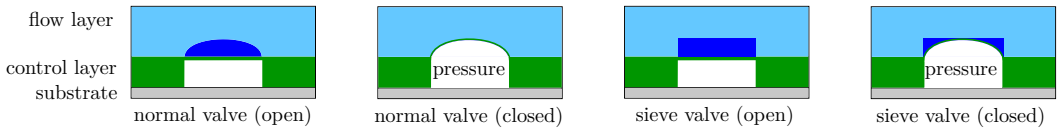

Figure 5.1: Difference between a normal valve and a sieve valve.

different from normal valves that can fully block fluid transportation, when a sieve valve is closed, there remains a gap that is small enough for tiny particles and fluids to flow away, but large enough to stop large particles such as magnetic beads bound by molecules or cells. In this manner, targeted large particles are concentrated or collected easily.

We introduced washing operations and sieve valves to the design automation field in 2016, by proposing a scheduling and operation-device mapping methodology for designs involving sieve valves[14]. Unlike typical operation-device mapping concepts, where operations are mapped to devices that occupy exclusive chip area, washing operations are supposed to be mapped to sieve valves, which hardly require any chip area and are always integrated inside devices, or near the entrance/exit of devices. Since a washing operation always accompanies a mixing operation, we modeled washing operations as pre-washing or post-washing *behaviors*, which were treated as special requirements of mixing operations. The synthesis results were generated by solving an ILP model, which modeled the general characteristics of continuous-flow microfluidics, as well as the special execution requirements.

Besides washing behavior, there could be other requirements for the execution of an operation, which used to be overlooked in conventional approaches for assay protocol interpretation. We introduced these execution limitations in our work and added corresponding constraints to our model: 1. Immediate execution: some operations are time-sensitive and must be executed immediately after

the completion of their parents. 2. Mutual exclusion: some operations must be mapped to different devices to avoid fluid contamination. 3. Parallel execution: some operations must be executed in parallel for fair comparison.

In the following, we describe our proposed method in detail. Similar descriptions can also be found in our published paper[14].

5.1 Mathematical model for washing behavior and specific execution limitations

In order to obtain an optimal bio-chip design for a bio-assay, we introduce an integer-linear-programming (ILP) model to simulate the assay process. According to the number of operations in an assay, we set up in our model a number of devices, some of which will be removed if the synthesis result shows that no operations are executed in them. The number of such devices is adjustable and represents the maximum number of devices that are allowed to be integrated in the final design. For convenience, we index all the operations and devices. We apply similar constraints as those mentioned in Section 4.1 to model the general characteristics of bio-assay execution: operation binding, operation duration, operation dependency, and non-interfering operation, and further consider the washing bahaviors as well as specific execution limitations.

1. *Washing behavior*

   We introduce two binary variables $pre\_w_{i,j}$ and $post\_w_{i,j}$, to represent whether a sieve valve needs to be integrated before or after a device i mapped by operation j. And we introduce the following constraints on each operation that requires a pre- or post-washing behavior, to make sure that such operations can only be bound to devices connected with corresponding sieve

valves:

$$if\ j\ requires\ a\ pre-washing\ behavior,$$
$$d\_o_{i,j} = pre\_w_{i,j}, \quad \forall i \in DEV, \tag{5.1}$$
$$if\ j\ requires\ a\ post-washing\ behavior,$$
$$d\_o_{i,j} = post\_w_{i,j}, \quad \forall i \in DEV, \tag{5.2}$$

where $d\_o_{i,j}$ is an operation-device-mapping variable representing whether operation j is bound to device i and DEV is the index set of all devices. The above constraints mean that for an operation j requiring pre-/post-washing bahavior, the value of $d\_o_{i,j}$ and $pre/post\_w_{i,j}$ must stay the same.

Now we can obtain the number of sieve valves that need to be integrated in the same manner as we obtain the number of devices: we first set up a variable sum_s to represent the number of sieve valves, then introduce the following constraints with auxiliary variables $act\_svpre/post_i$ on each device:

$$pre\_w_{i,j} - act\_svpre_i \leq 0, \quad \forall j \in preO, \tag{5.3}$$
$$post\_w_{i,j} - act\_svpost_i \leq 0, \quad \forall j \in postO, \tag{5.4}$$

where $pre/postO$ are the index sets of operations requiring pre-/post- washing behaviors. And we obtain sum_s as the sum of auxiliary variables:

$$sum_s = \sum_{i \in DEV} (act\_svpre_i + act\_svpost_i). \tag{5.5}$$

2. *Specific execution limitations*

   The execution of biochemical operations can be limited by different reagent properties and assay objectives, which requires synthesis adaption. Therefore, we modify our model further and provide a synthesis method considering three commonly seen limitations.

(a) *Immediate execution*

Sometimes, the transition time between sequential operations needs to be strictly controlled to prevent the reagents from overreaction. In these cases, a child operation is required to be performed immediately after the completion of its parent operation. Therefore, we introduce the following constraints on sequential operations requiring immediate execution:

$$t_b = t_a + dur_a + t_{trans}, \tag{5.6}$$

where time variable t_i represents the start time of operation i, b represents the child operation of a, dur_a represents the duration of operation a, and t_{trans} is a constant representing the transportation time between devices or the preparation time between sequential operations. This constraint means that when operation a is finished, operation b will start within an experimenter-definable transition time.

(b) *Mutual exclusion*

Some biochemical operations are mutually exclusive, since the contamination of their reagents may cause serious errors to the assay. Therefore, these operations are supposed to be bound to two different devices, which can be modeled by introducing the following constraint on each device:

$$d\_o_{i,a} + d\_o_{i,b} \leq 1, \tag{5.7}$$

where a, b are operations with mutual exclusion, which are prevented by this constraint from being bound to the same device i.

(c) *Parallel execution*

Another commonly seen requirement in bioassays is the execution of replicate operations, some of which are performed as control group for reference. In order to provide a fair environment for these operations, they are usually performed in parallel in different devices. Since time-overlapped operations have been prevented from being bound to the same device, we only need to ensure that these operations start at the

same time:

$$t_a = t_b, \tag{5.8}$$

where a, b are operations requiring parallel execution.

3. *Objective*

   The complete ILP model can be formulated as:

$$\text{Minimize:} \quad t_e \cdot C_{t_e} + sum_d \cdot C_{sum_d} + sum_s \cdot C_{sum_s}, \tag{5.9}$$

$$\text{Subject to:} \quad \text{constraints (4.1)–(4.6) and (5.1)–(5.8)}, \tag{5.10}$$

   where t_e represents the duration of the whole assay, sum_d represents the number of devices that need to be integrated in the chip, and C_{t_e}, C_{sum_d}, and C_{sum_s} are adjustable constants, which helps to control the weight of time and area cost.

5.2 Experimental results

We implemented the proposed synthesis in C++ on a computer with a 2.67 GHz CPU. The ILP model was solved by the ILP solver Gurobi[10]. To demonstrate the effectiveness of our method, we applied our method to synthesize four bioassays[6,51,52,55].

The first test case is a ChIP assay[51] including 12 operations with 7 washing behaviors. As shown in Figure 2.3, operation 4 needs to be executed directely after operation 3 and operation $5, 6, 7, 8$ are supposed to be executed in parallel. The second test case is a kinase activity assay[6]. This case includes 22 operations with 22 washing behaviors, two operations thereof are mutually exclusive and should not be executed in the same device. The third test case is another ChIP

assay with more IPs[52]. 28 operations with 23 washing behaviors are included in this assay, thereof 12 operations are supposed to be executed in different devices in parallel. The fourth test case is a 20-single-cell mRNA-to-cDNA synthesis assay[55]. This is a big test case including 80 operations with 60 washing behaviors and correspondingly more execution limitations. If the durations of some assay operations are not specified in the protocols, we arbitrarily assign their values with the same value of the other operations in the assay.

Since the traditional method does not consider the integration of sieve valves and the above mentioned execution limitations, its synthesis result has to be refined to fit the bioassay requirements:

1. We integrate sieve valves before or after devices, which are bound by mixing operations requiring pre- or post-washing behaviors according to the binding result.

2. We refine the scheduling result by taking the execution time of washing behaviors into account.

3. We analyze the synthesis result further, to treat the violations of execution limitations.

- For violations of immediate execution, in order not to mess up the scheduling result, we remap the parent operation to an additional device and reschedule it to make it complete right before the start of its immediately to be executed child operation.

- For violations of mutual exclusion, we remap some operations to additional devices, to ensure that all operations with mutual exclusion are mapped to different devices.

- For violations of parallel execution, suppose that the set of operations which need to be executed in parallel is P, and op_l is the operation with the

latest start time in P. We make op_l keep its original status and remap the other operations in P to additional devices and reschedule them to make them start simultaneously with op_l.

In the remapping process, we first check whether there is an additional device which is not occupied during the execution time of the to be remapped operation. If there is, we map the operation to this free device to save the area cost. If there isn't any free device available, we add one more additional device to the design and map the operation to this new device.

In the rescheduling process, if operation a is rescheduled, the schedule of its succeeding operations and the schedule of the operations that share the same device with a and start later than a will also be influenced. Therefore, we reschedule these operations as well.

Table 5.1 shows the experimental results of our method and the traditional method. Two groups of data are provided with different emphases on model objective as: area-cost-sensitive and execution-time-sensitive as shown in each row. The meaning of the columns is as follows:

$\#op(\#w)$ **:** the number of operations and washing behaviors.

*emp.***:** the emphasis on model objective.

$\#vio$**:** the numbers of violations of execution limitations.

$\#d_o + \#d_a$**:** the number of devices in the original result applying traditional method and the number of additional devices by refinement.

$\#s_o + \#s_a$**:** the number of sieve valves integrated in the devices in the original result applying traditional method and in the additional devices by refinement.

T_e**:** the total assay execution time.

Table 5.1: Result comparison between traditional synthesis and our synthesis under different emphases.[14]

			Traditional Synthesis Method					Our Synthesis Method			
	$\#op(\#w)$	*emp.*	$\#vio$	$\#d_o + \#d_a$	$\#s_o + \#s_a$	T_e	T_r	$\#d$	$\#s$	T_e	T_r
ChIP[51] (4 parallel IPs)	12(7)	area	7	2+3	1+6	378	3.074	5	6	301	1.926
		time	8	4+3	1+6	321	5.059	7	6	235	4.057
kinase activity[6]	22(22)	area	1	1+1	2+2	755	30.489	2	3	835	30.263
		time	1	3+1	3+2	360	0.759	4	6	274	1.026
ChIP[52] (16 parallel IPs)	28(23)	area	11	2+11	1+22	1775	30.475	13	22	565	30.486
		time	24	6+11	1+22	387	5.541	16	22	288	5.434
mRNA-to-cDNA synthesis[55]	80(60)	area	48	2+38	2+38	1285	103.040	21	22	1780	101.424
		time	57	15+38	12+38	493	113.135	50	46	88	100.989

T_r**:** the program runtime.

$\#d$**:** the number of devices in the result applying our method.

$\#s$**:** the number of sieve valves in the result applying our method.

As shown in Table 5.1, since the traditional method hasn't considered all the necessary constraints during the optimization process, its original synthesis results may bring about numerous violations of execution limitations as shown in column $\#vio$, which will prevent these results from being realized as practical designs. Even though we have refined these results and make them meet the requirements of the test cases, the refined results are still evidently outperformed by our method.

Under consideration of washing behaviors and execution limitations, our method maximizes the utilization of devices and sieve valves. Within similar program runtime, our method provides results with less area cost (fewer devices or fewer sieve valves) in all four test cases under area-cost-sensitive setting, and shortens the assay execution time under time-cost-sensitive setting. When dealing with the biggest test case, the area cost is nearly halved and the time cost is cut to less than 1/5 under corresponding settings.

As a conclusion, we propose the first high-level synthesis method taking the sieve valve into consideration with the chip area cost and the assay execution time as the optimization objectives. A key contribution of this work was that we had taken a closer look to the biological field, and thus modeled the scheduling and operation-device mapping problem in a realistic manner. High-level synthesis for continuous-flow microfluidics is still in its early stage. Compared with developing new algorithms, it is more important to understand the demands of chip designers.

6. Synthesis for Reconfigurable Microfluidics

Classical continuous-flow microfluidics consists of dedicated devices and channels, and even a slight change of operation protocols may involve the re-design of the whole chip. The demand for reconfigurable microfluidic chips always exists[35]. One of the most practical approach for reconfigurable continuous-flow microfluidics was proposed by Fidalgo in 2011[7]. Fidalgo proposed and manufactured a general-purpose software-programmable chip, which had a matrix-shaped valve structure. By controlling the actuation of valves, devices and transportation channels could be formed at any location on the chip, which enabled the adjustment of assay protocols without hardware modification. However, the matrix-shaped design based on a large number of individual controllable valves, which resulted in much control effort as well as the concern about the yield, and was thus hard to be implemented for large-scale design.

Based on the proposed reconfigurable matrix-shaped valve structure, I provided a design-automation solution that synthesizes chips with reduced number of fabricated valves, and balanced valve actuation to alleviate reliability concerns[42,44]. Reliability is one of the key concerns for valve-based structures, since

valves may under the danger of defect after thousands of actuations[1]. The most vulnerable valves are valves for peristalsis, which need to be actuated frequently during mixing operations. Our method addressed this concern by changing the roles of valves during the execution process, and thus balanced the valve actuations.

In general, our method took the scheduling results and dependency graph of an assay as inputs, and synthesized chips that supported microfluidic devices of different volumes and orientations. We also routed the transportation channels among devices and chip ports, and proposed *in-situ* on-chip storages to improve the resource usage and execution efficiency.

In the following, we describe our proposed method in detail. Similar descriptions can also be found in our published papers[42,44].

6.1 Valve-centered Architecture

The idea of the valve-centered architecture is from a valve matrix proposed and manufactured by Fidalgo[7], in which valves are arranged regularly and every component including flow channels in the chip is completely constructed by valves, and the basic unit of a flow channel is a chamber encircled by four valves. Therefore, this valve matrix is programmable just like the electrode matrix in digital biochips. However, the number of valves implemented in the chip can be very large, which leads to much control effort. In this paper, we transform that valve matrix into a valve-centered architecture with virtual valves.

In the valve-centered architecture, virtual valves are arranged regularly. A 4×4 example in a coordinate system is shown in Figure 6.1(a). These valves are *virtual* because some of them may not be manufactured as real valves, but

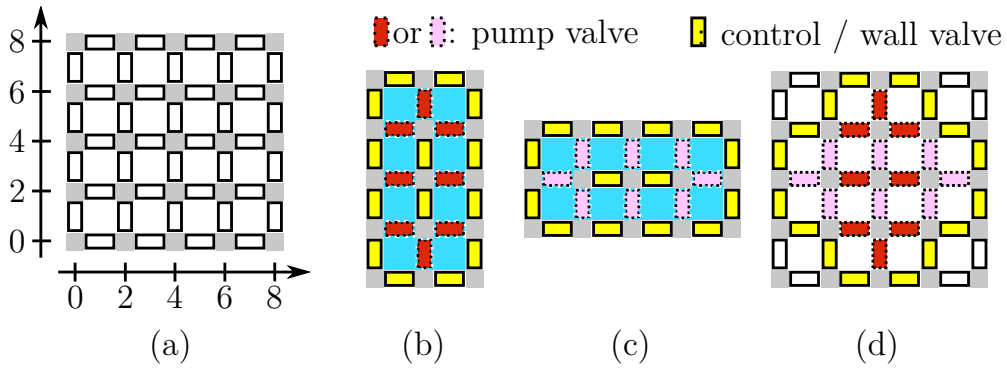

Figure 6.1: (a) A 4×4 valve-centered architecture (b) A 2×4 dynamic mixer. (c) A 4×2 dynamic mixer. (d) Dynamic mixers of different orientations sharing the same area.[44]

removed after synthesis. The virtual valves can be used as wall valves to construct the boundary walls of devices, so that devices can be formed and split up on request dynamically during the biochemical assay. We call such kind of devices *dynamic devices*.

In the valve-centered architecture, different dynamic devices can share the same area without making any valve play the role as pump valve twice so that the valves may not be worn out so fast. For example, two 2×4 mixers with different orientations as shown in Figure 6.1(b)(c) can be generated in the same region at different time as shown in Figure 6.1(d): though the two mixers overlap with each other, their pump valves are completely different.

6.2 Dynamic Device Mapping

To generate dynamic devices at the best locations in the valve-centered architecture and thus achieve the most reliable designs, we propose an integer linear programming (ILP) model to accurately model valve actuation brought by constructing dynamic devices. In this model, instead of modeling all actuation activities, we only model the actuation activities for peristalsis, since pump valves dominate

the valve actuation problem. To determine the location, shape, and orientation of each dynamic device, we introduce a binary variable $s_{x,y,k,i}$ as *selection variable*. (x, y) is the left-bottom corner coordinate of a device to represent its location, for example, $(0, 0)$, $(2, 0)$, $(0, 2)$, $(2, 2)$ as shown in Figure 6.2(b)(c)(d)(e); k represents the index of a device type, i.e., the shape and the orientation of the device, such as 1 for 3×3, 2 for 2×4, and 3 for 4×2; i is the index for the i_{th} operation. When a selection variable $s_{x,y,k,i}$ is set to 1, it means that the i_{th} operation is mapped to a device of type k at the location (x, y). Since each operation can only be mapped to a single device, we introduce the following constraint

$$\sum_{x,y,k} s_{x,y,k,i} = 1, \qquad \forall i \leq |O| \tag{6.1}$$

where O is the set of all operations in the assay.

Each time when an operation is mapped to a dynamic mixer, some virtual valves related to this mixer will work as pump valves. With location, shape, and orientation information of a device, the coordinates of these temporary pump valves are ascertained. We represent the number of valve actuations for peristalsis of each virtual valve by an integer variable $v_{x,y}$ and calculate it as

$$v_{x,y} = \sum_{x_p,y_p,k,i} p_i s_{x_p,y_p,k,i}, \qquad \forall (x, y) \in C, \qquad \forall s_{x_p,y_p,k,i} \in S \tag{6.2}$$

where p_i is a constant representing the number of actuations for a pump valve to perform the i_{th} mixing operation, C is the set of all coordinates, and S is a set containing all selection variables $s_{x_p,y_p,k,i}$ that satisfy the following condition: when $s_{x_p,y_p,k,i}$ is set to 1, a k-type mixer will be generated with left-bottom corner at (x_p, y_p) to perform the i_{th} mixing opration, and the virtual valve at (x, y) will

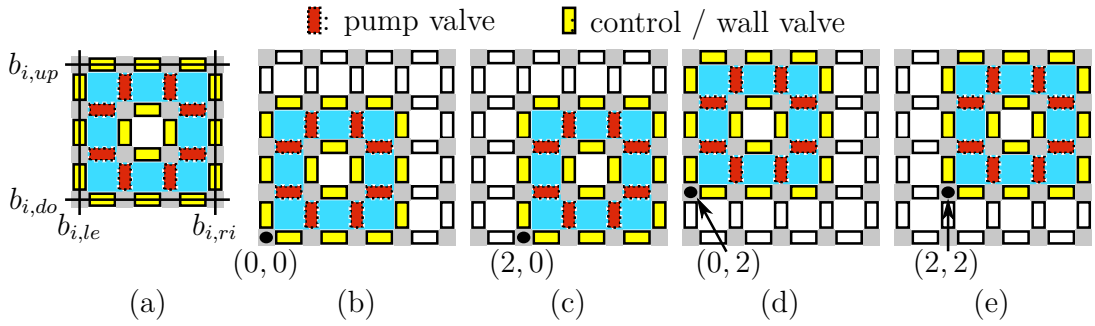

Figure 6.2: (a) A 3×3 dynamic mixer with 8-unit volume. (b)(c)(d)(e) Four possible locations to place a 3×3 mixer.[44]

work as one of its pump valves.

To avoid generating different devices in the same area at the same time, we introduce four more integer variables as $b_{i,le}$, $b_{i,ri}$, $b_{i,up}$, and $b_{i,do}$. As shown in Figure 6.2(a), $b_{i,le}$, $b_{i,ri}$, $b_{i,up}$, $b_{i,do}$ represent the coordinates of all wall valves, which build the boundaries of the dynamic device that the i_{th} operation is mapped to. By using these variables, the non-overlapping constraints for two devices mapped by operations i_1 and i_2 can be modeled as

$$(b_{i_1,ri} \leq b_{i_2,le}) \vee (b_{i_1,le} \geq b_{i_2,ri}) \vee (b_{i_1,up} \leq b_{i_2,do}) \vee (b_{i_1,do} \geq b_{i_2,up}) \tag{6.3}$$

which can be transformed into linear form as

$$b_{i1,ri} \leq b_{i2,le} + c_1 M, \tag{6.4}$$

$$b_{i1,le} \geq b_{i2,ri} - c_2 M, \tag{6.5}$$

$$b_{i1,up} \leq b_{i2,lo} + c_3 M, \tag{6.6}$$

$$b_{i1,lo} \geq b_{i2,up} - c_4 M, \tag{6.7}$$

$$c_1 + c_2 + c_3 + c_4 = 3 \tag{6.8}$$

in which c_1, c_2, c_3, c_4 are auxiliary binary variables, and M is a very large constant. From constraint (6.4) to (6.7), when one of c_k, $k \in \{1, 2, 3, 4\}$ is set to 1, the corresponding inequation becomes trivial. However, with constraint (6.8), one of the elements in the set $\{c_1, c_2, c_3, c_4\}$ must be set to 0, so that at least one of the four non-overlapping conditions can be successfully fulfilled.

With the constraints mentioned above, we build an ILP model to minimize the highest $v_{x,y}$, which is the largest number of actuations of those valves for peristalsis. We bound this number by an integer variable w with the following constraint

$$v_{x,y} \leq w, \qquad \forall (x, y) \in C \tag{6.9}$$

and the whole model can be described as

$$\text{Minimize: } w \tag{6.10}$$

$$\text{Subject to: constraints } (6.1) - (6.2), (6.4) - (6.9) \tag{6.11}$$

6.3 In Situ On-chip Storages

In a biochemical assay, the product of a preceding operation is usually the input of a later operation. We call the preceding operation *parent operation* of the later operation, and the later operation *child operation* of the preceding operation. Correspondingly, the device performing the parent operation is called the *parent device* of the device performing the child operation, and the device performing the child operation is called the *child device* of the device performing the parent operation. Because an operation can only start after all its inputs are ready, the products of preceding operations need to be stored. A traditional practice is to build some dedicated storages, which need extra chip area and can cause transport

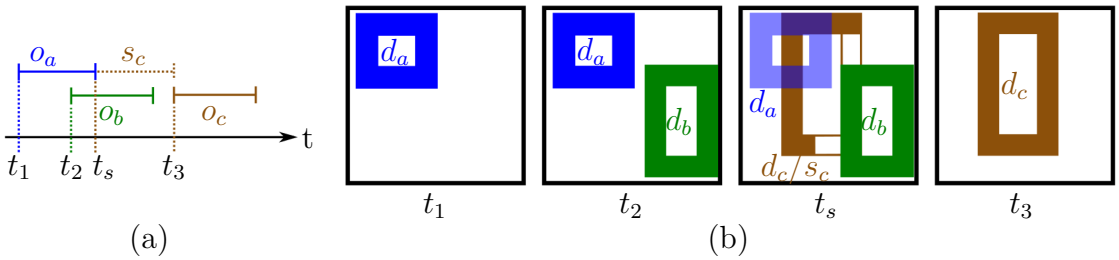

Figure 6.3: An example of an *in situ* on-chip storage s_c: (a) Scheduling result. (b) Chip snapshots at different time.[44]

delay. In our method, with the valve-centered architecture, we generate dynamic devices as *in situ* on-chip storages to store coming products, so that chip area and transportation time can be saved.

An example is shown in Figure 6.3, in which the scheduling result is drawn as a Gantt Chart, and the dynamic mixers are simplified and drawn as the circulation flows that they contain. o_a, o_b, and o_c are mixing-operations, in which o_c takes the products of o_a and o_b as its inputs and therefore o_c is the child operation of o_a and o_b. d_a, d_b, and d_c are dynamic devices for o_a, o_b, and o_c. s_c is an *in situ* on-chip storage that will be transformed into d_c directly after collecting all inputs, and thus save the transportation effort.

At time t_s, o_a is completed and thus the valves which have constructed d_a can be treated as free valves, so that we can build s_c by using some of these valves to store the product of o_a immediately. Since s_c only contains the product of o_a at time t_s, there is still some free space inside it. In our method, we take advantage of those free spaces by allowing them to overlap with their parent devices. In this example, o_b is in process at time t_s. Therefore, s_c only occupies part of the later d_c until o_b is completed at time t_3. Then s_c is turned to d_c by using the free valves of the former d_b, and the product of o_b can also conveniently be led to d_c for the coming operation.

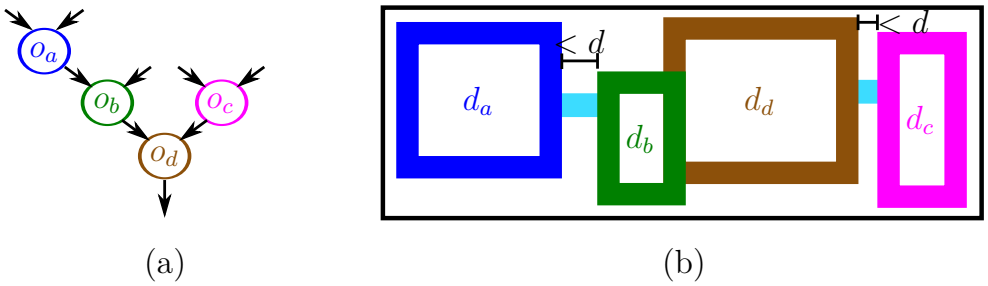

Figure 6.4: An example of routing-convenient dynamic device mapping: (a) Sequencing graph. (b) Device locations.[44]

To implement this special overlapping permission to our ILP model, we only need to add an auxiliary binary variable c5 to constraint (6.8)

$$c_1 + c_2 + c_3 + c_4 = 3 + c_5. \tag{6.12}$$

If c_5 is set to 1, c_1, c_2, c_3 and c_4 must all be 1, which permits the overlapping between two devices. But if we do not want this overlapping to happen, we can set c_5 to 0, so that the meaning of this constraint will be the same as constraint (6.8).

6.4 Routing-convenient Mapping

Our dynamic device mapping also guarantees the transportation paths between parent and child devices, which brings convenience to routing. When we map two sequential operations to two different devices, we prefer to build a direct connection between these devices to save the transportation effort. In order to do that, we introduce a constant d, which is the minimum dimension of all devices, as the maximum distance between a parent device and its child device. This distance

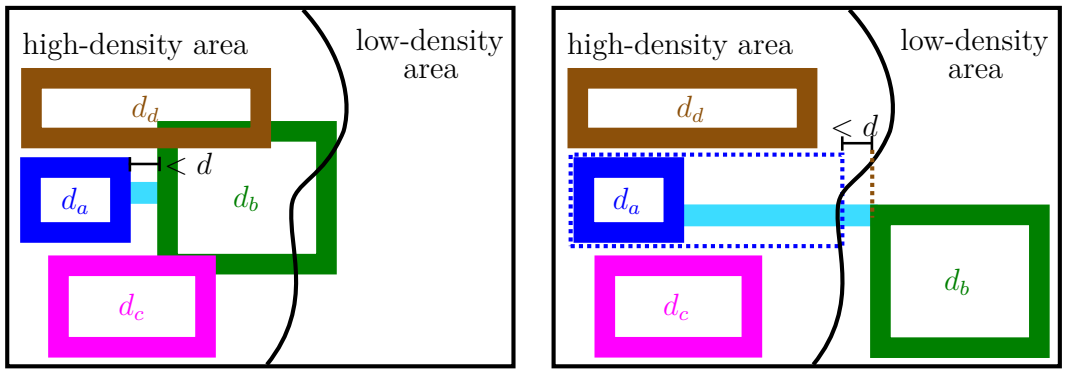

Figure 6.5: Device location of d_b: (a) Without virtual boundaries. (b) With virtual boundaries.[44]

limit can be introduced to our model by adding four more constraints:

$$b_{i1,ri} \geq b_{i2,le} - d, \tag{6.13}$$

$$b_{i1,le} \leq b_{i2,ri} + d, \tag{6.14}$$

$$b_{i1,up} \geq b_{i2,lo} - d, \tag{6.15}$$

$$b_{i1,lo} \leq b_{i2,up} + d \tag{6.16}$$

where i_1 is the parent operation of i_2.

These constraints ensure that the distance between a parent device and its child device is short enough to prevent other devices from being inserted between them and thus obstructing their connection path. For example, as shown in Figure 6.4(a), suppose o_a is the parent operation of o_b, and o_d is the child operation of o_b and o_c. If these operations are mapped to different devices, by controlling their device locations, direct connections can be easily built between parent and child devices as shown in Figure 6.4(b).

However, with a strict distance control, the number of potential device lo-

cations will be remarkably reduced, since a child device has to be put next to its parent device. An example is shown in Figure 6.5(a), the left part of the chip is crammed with devices while the other part is left unused. This limitation may keep us from finding an optimal solution and lengthen the optimization process.

Since our target is to build direct connections between parent and child devices, but not necessarily to put them close to each other, we introduce *virtual boundaries* to refine our distance controlling method. The virtual boundaries of a device circle a *virtual area* which is larger than or equal to the real size of this device, which can be introduced to our model by adding following constraints:

$$b'_{i,ri} \geq b_{i,ri}, \tag{6.17}$$

$$b'_{i,le} \leq b_{i,le}, \tag{6.18}$$

$$b'_{i,up} \geq b_{i,up}, \tag{6.19}$$

$$b'_{i,lo} \leq b_{i,lo} \tag{6.20}$$

where $b'_{i,ri}$, $b'_{i,le}$, $b'_{i,up}$, and $b'_{i,lo}$ are the virtual boundaries of the dynamic device that the i_{th} operation is mapped to.

Instead of controlling the distance between exact device locations, we control the distance between virtual areas, which can be easily introduced to our model by replacing the boundary variables in constraints (6.13)-(6.16) with virtual boundary variables. As shown in Figure 6.5(b), d_a and d_b are no longer forced to be put together. Therefore, d_a can be located in the low-density area of the chip.

We have also modified the overlapping rule of devices with our virtual-area concept by replacing the boundary variables in constraints (6.4)-(6.7) with virtual

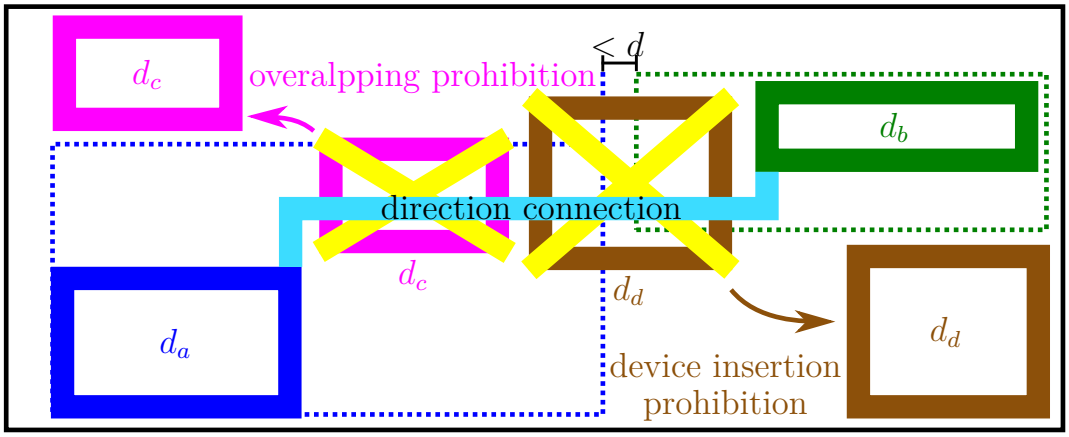

Figure 6.6: Guarantee of direct connection from d_a to d_b.[44]

boundary variables. As shown in Figure 6.6, by prohibiting the overlapping among virtual areas instead of the exact device locations, we prevent the direct connection between parent and child devices from being obstructed by other devices.

Since the occupancy rate of chip area varies during an assay process according to the assay sequencing graph and the scheduling result, virtual boundaries can be applied especially when the occupancy rate of chip area is low so that the virtual area is not an issue for area competition among devices. As a conclusion, the introduction of virtual boundaries provides us more flexibility of locating devices and thus allows us to maximize the utilization of chip area and balance the valve actuations even further.

6.5 Assurance of Fluid Paths to Chip Boundaries

In order to transport waste, samples, reagents, and final products, devices need to be connected with chip ports. Since our valve-centered architecture is implemented in a matrix-shaped biochip[7], in which chip ports are all located at the chip boundaries, we propose a method to assure the fluid paths from a dynamic

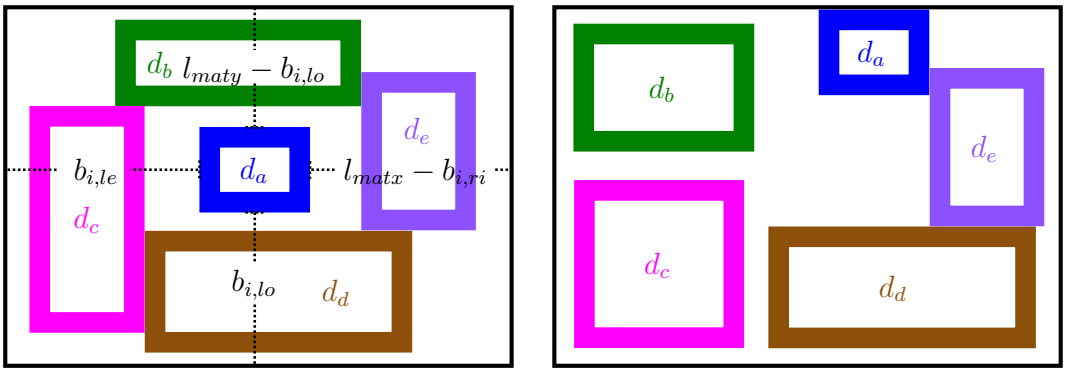

Figure 6.7: An example of fluid path assurance for d_a to chip boundaries: (a) d_a is freely placed. (b) d_a is closer to chip boundries.[44]

device to chip boundaries.

When a device lies in the inner part of a chip and is completely encircled by other devices as shown in Figure 6.7(a) after the dynamic device mapping, the path between a device and chip boundaries can be blocked. Therefore, we break the encirclement by adding a new objective in our ILP model to draw this certain device closer to the corner of the chip, and perform the dynamic device mapping again.

Before introducing the objective, we first define the distance between a device and chip corners. As shown in Figure 6.7(a), the distance between d_a and chip corners is decided by the horizontal and vertical distance between d_a and the chip boundaries. This can be introduced to our model by adding following

constraints:

$$l_{i,1} \geq b_{i,le} - q_1 M, \tag{6.21}$$

$$l_{i,1} \geq l_{matx} - b_{i,ri} - q_2 M, \tag{6.22}$$

$$l_{i,2} \geq b_{i,lo} - q_3 M, \tag{6.23}$$

$$l_{i,2} \geq l_{maty} - b_{i,up} - q_4 M, \tag{6.24}$$

$$q_1 + q_2 = 1, \tag{6.25}$$

$$q_3 + q_4 = 1 \tag{6.26}$$

in which $l_{i,1}$, $l_{i,2}$ are the horizontal and vertical distance between a device and its nearest chip corner, l_{matx}, l_{maty} are the horizontal and vertical dimensions of the chip, q_1, q_2, q_3, q_4 are auxiliary binary variables and M is a very large constant.

When one of q_k, $k \in \{1, 2, 3, 4\}$ is set to 1, the corresponding inequation becomes trivial. Taking Figure 6.7(a) as an example, the set of horizontal distances between d_a and chip boundaries is $\{b_{i,le}, l_{matx} - b_{i,ri}\}$, and the set of vertical distances between d_a and chip boundaries is $\{b_{i,lo}, l_{maty} - b_{i,up}\}$. Constraints (6.21)(6.22) ensure that we will choose exactly one value from each set to control the distance between d_a and its nearest chip corner. For the sake of model reduction, we represent this distance by $l_{i,1} + l_{i,2}$, instead of $\sqrt{l_{i,1}^2 + l_{i,2}^2}$ applying the Pythagorean theorem. Accordingly, we modify our optimization objective:

$$\text{Minimize: } w + \alpha f_i \times (l_{i,1} + l_{i,2}), \forall l_i \in S_d \tag{6.27}$$

where α is a constant coefficient, f_i is a weight factor which increases each time the connection path problem occurs to the i_{th} operation, and S_d is the set of operations whose device connection to chip boundaries are blocked.

With this modification, as shown in Figure 6.7(b), we can obtain a new mapping result. Based on this different request, the shape and location of our dynamic devices are adjusted so that d_a is drawn near the upper right chip corner and no longer encircled by other devices, which assures the fluid path between d_a and chip boundaries and thus chip ports.

6.6 Valve-actuation-aware Routing

After the dynamic device mapping process, we route the fluid paths in the chip. Our routing method takes valve actuations caused by path routing into consideration and thus further reduces the maximum number of valve actuations and the number of valves.

We apply Dijkstra's shortest path algorithm and construct the cost function according to valve actuations. In our valve-centered architecture, fluid paths can be divided to chambers formed by valves. As shown in Figure 6.8(a), A, B, C, and D are such chambers. By controlling the valves connected with these chambers, namely v_1, v_2, v_3, and v_4, we can control the direction of fluids and thus build different fluid paths.

Before we route a new fluid path, we record the number of current valve actuations of each valve and set this valve as the cost of this valve, and we set the initial cost of each chamber as infinity, as shown in Figure 6.8(b).

When the actuation of valve v is involved in forming a fluid path to a chamber C_H, the cost of v will be added with the cost of the chamber as the front of the path. We define the sum of the costs as s and compare it with the cost of chamber C_H. If s is smaller, we update the cost of chamber C_H with s. Our target is to find the lowest cost of each chamber that we want to reach to from a

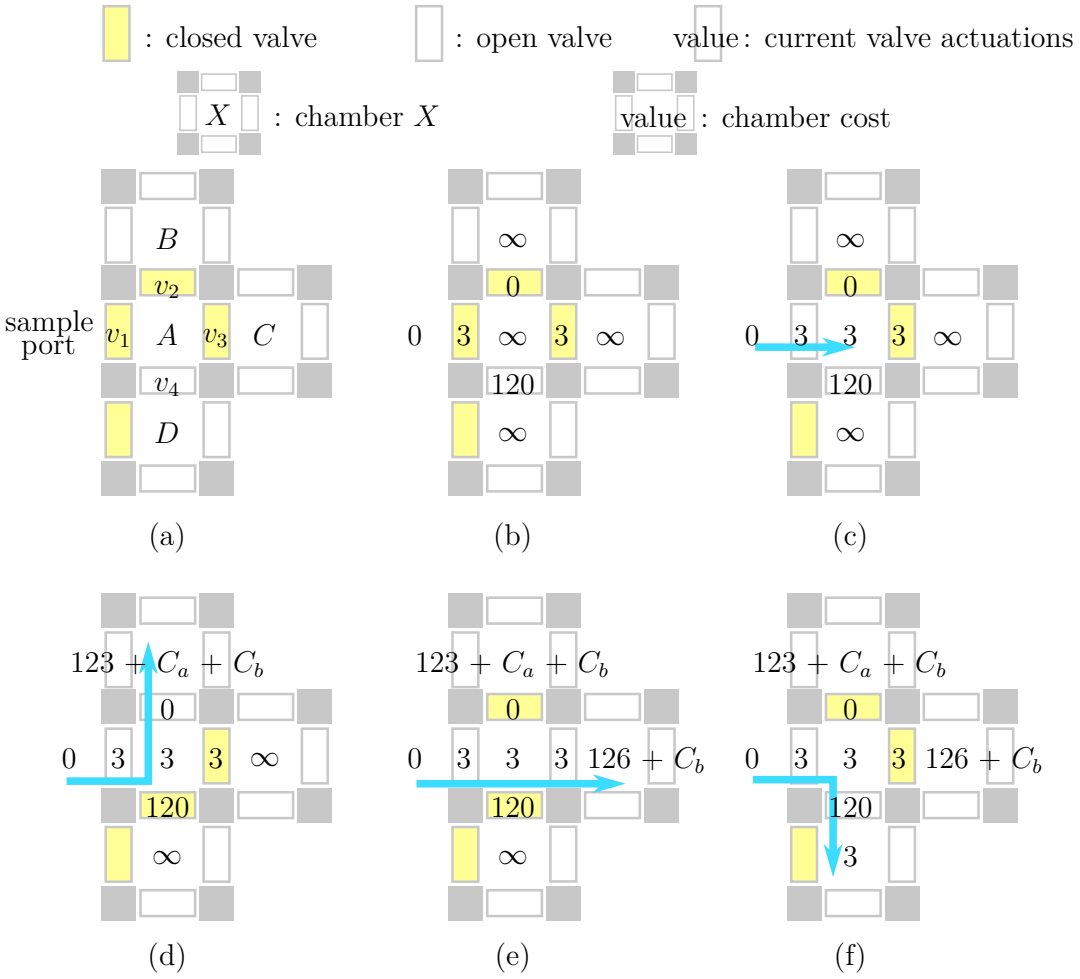

Figure 6.8: Update of chamber costs: (a) Target chambers. (b) Initial costs. (c) For chamber A. (d) For chamber B. (e) For chamber C. (f) For chamber D.[44]

starting point, and thus deciding the routing path with back-tracing. As shown in Figure 6.8(b), the cost of the starting sample port is set to 0. In order to reach chamber A, we add the cost of valve v_1 with 0 and get a new value 3, which is smaller than infinity, and thus replace infinity as the new cost of chamber A as shown in Figure 6.8(c).

In our example, v_4 has been used as pump valve for multiple times and thus actuated for 120 times. We suppose that 120 is exactly the largest number of valve actuations in the chip. On the other hand, v_2 has not been actuated yet and may be removed at the end of the entire synthesis. Therefore, when the actuation of v_4 and v_2 is involved in forming a routing path, we will either worsen our former optimization result, or we will need to manufacture an extra new valve. In order to reduce these actuations, we set extra cost for actuating these valves. As shown in Figure 6.8(d), we set extra cost C_a to v_2 which is never used and extra cost C_b to v_4 which has the highest number of actuations. When we want to reach chamber B from chamber A, we need to actuate v_2 and v_4. Therefore, the cost of chamber B will be updated with the sum of costs of v_2, v_4, and A, which is $123 + C_a + C_b$, since this sum is less than infinity. Correspondingly, the new cost of chamber C is $126 + C_b$ and the new cost of chamber D is 3, as shown in Figure 6.8(e)(f).

This cost function can be represented as:

$$\begin{aligned} \text{if} \quad & c_N < c_C + s_{v_i} a_{v_i} \\ & \qquad + (1 - u_{v_i}) C_a + m_{v_i} C_b, \forall v_i \in S_C, \\ \text{then} \quad & c_N = c_C + s_{v_i} a_{v_i} \\ & \qquad + (1 - u_{v_i}) C_a + m_{v_i} C_b, \forall v_i \in S_C \end{aligned} \tag{6.28}$$

where c_N is the cost of the to-be-reached chamber N, c_C is the cost of the last passed-by chamber C, s_{v_i} is a binary variable indicating whether the actuation of valve v_i is involved in forming a fluid path from C to N, a_{v_i} is the number of actuation of v_i, u_{v_i} is a binary variable indicating whether v_i has ever been actuated, m_{v_i} is a binary variable indicating whether v_i is the valve with the highest number of actuations, and S_C is a set containing the valves encircling chamber C.

According to the assay schedule, each time when we need a new routing path, we accurately record the current valve status to decide which valve actuations should be involved in forming this new routing path. Based on these information, we apply our above mentioned method to get a routing solution. But since the assay is in progress, a valve with currently fewer actuations may also serve as a frequently actuated pump valve later. Therefore, we apply a rip-up and reroute method for several iterations based on the former routing results to revise our routing solution.

Suppose that we want to revise the costs of chambers A, B, C and D as shown in Figure 6.9(a). Valve v_1, v_2, v_3 and v_4 are currently actuated for 3, 2, 5 and 40 times respectively. As shown in Figure 6.9(b), we know from the former routing results that in the last iteration, from the current time till the end of the assay, v_2 will be actuated 80 more times. Therefore, we revise the cost of v_2 by adding 80 to it. Similarly, we also add the cost of v_3 with 3. We then get the new costs of chambers A, B, C, D as shown in Figure 6.9(c), which provides us a more comprehensive solution.

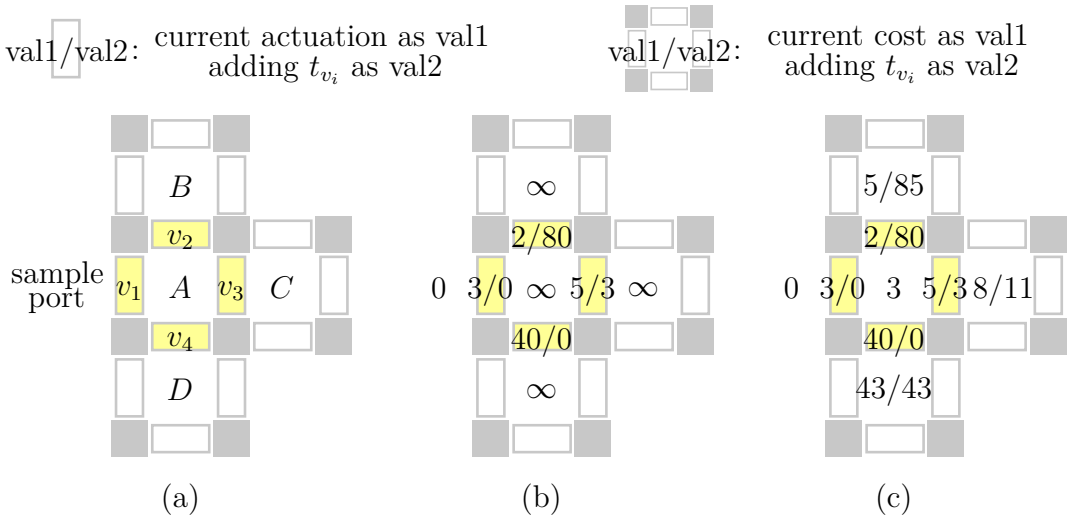

Figure 6.9: Chamber costs applying rip-up and reroute: (a) Target chambers. (b) Initial costs. (c) Updated chamber costs.[44]

The revised cost function can be formulated as follows:

$$\begin{aligned} \text{if} \quad & c_n < c_c + s_{v_i} \times a_{v_i} \\ & + (1 - u_{v_i})C_a + m_{v_i}C_b \\ & + t_{v_i}, \forall v_i \in S_c, \\ \text{then} \quad & c_n = c_c + s_{v_i} \times a_{v_i} \\ & + (1 - u_{v_i})C_a + m_{v_i}C_b \\ & + t_{v_i}, \forall v_i \in S_c \end{aligned} \tag{6.29}$$

in which t_{v_i} indicates the extra actuations of v_i from the current time till the end of the assay in the last iteration. In this way, we maximize the utilization of existing valves with fewer actuations and thus also existing flow channels, which enables us to reduce the largest number of valve actuations and the sum of valves even further.

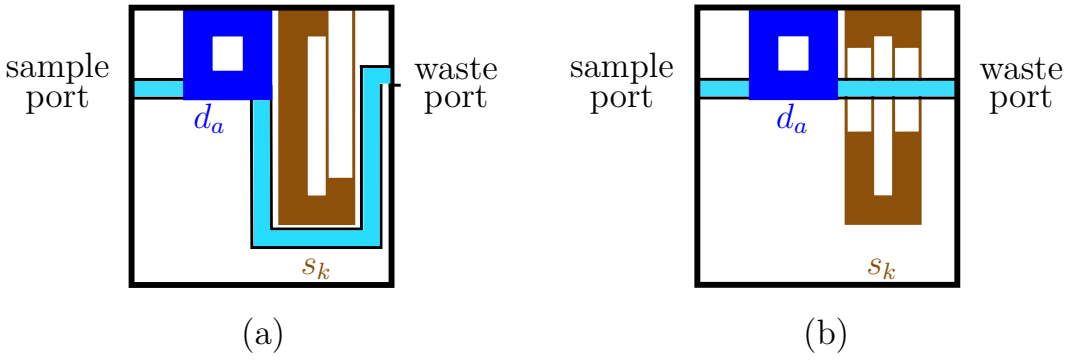

Figure 6.10: (a) The storage s_k is an obstacle for routing paths. (b) The storage s_k can be passed through by routing paths.[44]

6.7 Overall Algorithm

Algorithm 2 gives an overall view of our methods. We index the lines as Li, $i \in \mathbb{N}$, at the beginning of each line. After reading the program input as shown in L1 and building the data structure as shown in L2, we perform our dynamic device mapping by using an ILP model as shown in L3-L12 and then decide the routing paths as shown in L13-L26.

After we get our first dynamic device mapping results, we perform an area check in L6-L8 and a fluid-path-assurance check in L9-L11 to support the reliability of our method:

Our valve-role-changing concept brings us more options for overlapping. Besides the overlapping permission for *in situ* on-chip storages and parent devices as mentioned in Section 6.3, when a storage has enough free space, we also allow routing paths to pass through this storage as shown in Figure 6.10(b), thus saving the efforts for a long detour as shown in Figure 6.10(a). To make sure that overlapping only happens on the premise of enough free storage-space, we perform an area check as shown in L6-L8 and L19-L22. We then perform a fluid-path-

Algorithm 2: Reliability-aware synthesis.[44]

```
L1  Read sequencing graph and scheduling result.
L2  Build virtual valves in valve-centered architecture.
L3  # DynamicDeviceMapping
L4  repeat
L5      Build and solve ILP model for dynamic device mapping.
L6      if overlapping area of (storage s, device d) > free space of s then
L7          Forbid (s,d) from overlapping with each other.
L8      end
L9      if fluid paths from or to device d can not reach chip boundaries then
L10         Draw d closer to chip boundaries.
L11     end
L12 until feasible dynamic device mapping;
L13 # Routing
L14 for iteration ite = 1 to maxIte do
L15     Rip up all routed paths.
L16     for time t = 1 to maxT do
L17         forall the connections do
L18             Route a path with minimum cost.
L19             if overlapping area of (storage s, path p) > free space of s
                then
L20                 Forbid (s,p) from overlapping with each other.
L21                 Rip up p and reroute.
L22             end
L23         end
L24         Record the numbers of valve actuations.
L25     end
L26 end
L27 Remove non-actuated valves.
```

assurance check as mentioned in Section 6.5 as shown in L9-L11.

We route the fluid paths after dynamic device mapping. The valve-actuation-aware routing results will be revised by a rip-up and reroute method for several iterations as shown in L14-L26.

6.8 Experimental results

We implemented the reliability-aware synthesis in C++ on a computer with a 2.67 GHz CPU. The ILP model for dynamic device mapping was solved by the ILP solver Gurobi[10]. The applied device library applied is shown in Table 6.1, where *volume* indicates the number of chambers occupied by a device, *dimension* indicates the number of chambers in horizontal and vertical directions of this device, and *ratio* indicates the input ratios that are supported by the device.

In our method, we assume that mixing operations with the same input volume and ratios have the same duration regardless of the mixer dimensions, and this duration indicates the maximum duration in mixers of all dimensions. For example, suppose that a mixing operation o_a can be executed in either mixer m_1 or mixer m_2 (m_1 and m_2 only differ in dimensions), the execution time of o_a in m_1 is t_1 and the execution time of o_a in m_2 is t_2. If $t_1 > t_2$, we will specify t_1 as the duration of o_a in our method, regardless of whether m_1 or m_2 will finally be used. The proposed method provides a conservative execution of operations in different mixers, and it can be extended easily to handle different execution durations by describing the execution time of an operation in different mixers with a lookup table.

We take four test cases from widely used laboratory protocols[3,29]. For each test case we set up three different policies with indices as p_1, p_2, and p_3. As the

Table 6.1: Library of devices used in this work.[44]

Volume	4	6	8	8	10	10
Dimension	2×2	2×3	2×4	3×3	2×5	3×4
Ratio	1 : 1	1 : 2	1 : 1, 1 : 3	1 : 1, 1 : 3	1 : 4, 2 : 3	1 : 4, 2 : 3

policy index increases, we increase the number of mixers used in a traditional design, in which dedicated mixers, storages, and detectors are used. Correspondingly, we can obtain different scheduling results as the inputs for experiments. We compare the experimental results of our method proposed in the conference paper[42] and the extended version proposed in the journal paper[44] under two different settings, a conservative setting and an aggressive setting, along with the results of the optimal binding for the traditional designs in Table 6.2, in which the meaning of each column is:

$\#op$ **:** the number of operations and mixing operations thereof.

$Po.$ **:** the policy index.

$\#d$**:** the number of devices, including mixers and detectors.

$\#m_{4-6-8-10}$**:** the numbers of operations bound to the same mixers, with hyphens separating mixers of different sizes.

vs_t_{max}**:** the largest number of valve actuations applying the optimal binding for the traditional designs.

vs_{max}**:** the largest number of valve actuations and actuations for peristalsis thereof applying our methods.

$\#v$**:** the sum of used valves.

T**:** the program runtime.

In Table 6.2, column 8-10 show the results of applying the method in the conference version under conservative setting, column 11-13 show the results of applying the method in the journal version under conservative setting, column

Table 6.2: Comparison of the highest valve actuation times and the number of valves.[44]

			Optimal Binding for Traditional Designs				Conf. Cons.[42]			Jour. Cons.[44]			Conf. Aggr.[42]		Jour. Aggr.[44]		
Column Index	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	$\#op$	$Po.$	$\#d$	$\#m_{4-6-8-10}$	vs_t_{max}	$\#v$	vs_{max}	$\#v$	T	vs_{max}	$\#v$	T	vs_{max}	$\#v$	vs_{max}	$\#v$	T
PCR	15(7)	p1	3	1-0-4-2	160	83	45(40)	71	0.8	42(40)	77	0.8	35(30)	71	31(30)	62	0.9
		p2	4	1-0-(2,2)-2	80	99	45(40)	76	0.8	44(40)	71	0.7	34(30)	76	36(30)	62	0.8
		p3	6	1-0-(2,1,1)-(1,1)	80	131	43(40)	82	0.9	44(40)	82	0.9	31(30)	82	32(30)	76	1.1
Mixing Tree	37(18)	p1	4	2-4-5-7	280	108	93(80)	105	2.9	87(80)	109	2.4	46(42)	105	35(30)	105	94
		p2	5	2-4-5-(4,3)	200	124	93(80)	105	2.9	87(80)	109	2.3	46(42)	105	35(30)	105	94
		p3	6	2-4-(3,2)-(4,3)	160	140	90(80)	124	3.3	90(80)	122	3.3	60(50)	124	38(30)	120	21.5
Interpolating Dilution	71(35)	p1	7	5-9-9-(6,6)	360	178	145(120)	176	357.1	132(120)	163	30.1	72(65)	176	62(42)	173	0.5h
		p2	9	5-(5,4)-(5,4)-(6,6)	240	207	94(80)	207	87.8	94(80)	207	20.7	56(42)	207	38(32)	206	1h
		p3	10	5-(5,4)-(5,4)-(4,4,4)	200	225	92(80)	208	101.2	90(80)	206	108	56(50)	208	47(35)	209	1.5h
Exponential Dilution	103(47)	p1	10	6-(8,8)-(7,6)-(6,6)	320	241	135(120)	214	485.3	101(80)	224	774.6	75(75)	214	55(50)	216	0.5h
		p2	11	6-(6,5,5)-(7,6)-(6,6)	280	254	134(120)	255	488.9	103(80)	254	858	71(65)	255	48(40)	261	1h
		p3	12	6-(6,5,5)-(5,4,4)-(6,6)	240	268	99(80)	259	314.3	93(80)	259	957.6	58(40)	259	47(40)	253	1.5h

14-15 show the results of applying the method in the conference version under aggressive setting, and column 16-18 show the results of applying the method in the journal version under aggressive setting.

In the traditional designs, we assume there are 4 different sizes of mixers: 4, 6, 8, and 10. The 4-unit mixers with two ports can support 1:1 mixing operations, the 6-unit mixers with two ports can support 1:2 mixing operations, the 8-unit mixers with three ports can support 1:1 as well as 1:3 mixing operations, and the 10-unit mixers with three ports can support 2:3 as well as 1:4 mixing operations. Each design contains a storage to store products temporarily, and the number of cells in the storage is determined by the largest number of simultaneous accesses to the storage.

Each assay operation, according to the volume of its inputs, is assigned to a mixer with the required size. If there are multiple mixers with the same size, we apply an optimal binding regarding valve actuation by distributing operations to mixers as evenly as possible. Because the loadings on mixers with different sizes may vary considerably, we add one more mixer for each mixer type that is under the heaviest loading as the policy index increases to alleviate the heavy burden. For example, as shown in Table 6.2, in test case PCR policy 1, there are 3 mixers with different sizes. 1 mixing operation is bound to the 4-unit mixer, 4 mixing operations are bound to the 8-unit mixer, and 2 mixing operations are bound to the 10-unit mixer. Hence we add one more 8-unit mixer in policy 2, so that in the result of the optimal binding the 4 mixing operations can be evenly assigned to the two 8-unit mixers as 2 operations per mixer.

In our method, we first built a square matrix containing virtual valves based on the valve-centered architecture. In this matrix, the number of virtual valves is larger than 1.5 times the number of valves used in the traditional method,

and the total fluid volume of the matrix is larger than 2 times the highest total fluid volume of the operations that simultaneously work in the chip. This setting is arbitrary, but not harmful to the number of valves implemented at the end, because the non-actuated virtual valves are removed after the synthesis.

After constructing the matrix, we built and solved the model for dynamic device mapping and routed the sample paths. We calculated the largest numbers of valve actuations in vs_{max} in Table 6.2, which are close to the numbers of actuations for peristalsis thereof. This fact supports our modeling methodology where we only model actuation activities for peristalsis.

In our model, all valves passed by the circulation flow inside a dynamic mixer are regarded as pump valves. Though in our method we use more pump valves, so that theoretically the loading on each valve should be alleviated under the same efficiency, it is difficult to tell how many actuations are sufficient for a single mixing operation. Therefore, we provide both a conservative setting and an aggressive setting for comparison with the traditional method.

Under our first setting we assume that each pump valve is actuated 40 times for a single mixing operation, which is exactly the same as the setting for pump valve working in a dedicated mixer in the traditional method as a conservative comparison. vs_{max} in column 8 and column 11 show that even under this conservative setting, we still reduce the largest numbers of actuations by more than 50% compared with traditional method. By contrast, under our aggressive setting, we assume that the sum of actuations for peristalsis of a mixer is the same as that in the traditional method. For example, the sum of valve actuations for peristalsis of a dedicated mixer to perform a single mixing operation in a traditional design is $3\times40=120$, so we change the number of actuations of each valve in our dynamic mixer using 8 pump valves to 15 since $8\times15=120$. As shown in vs_{max} in column

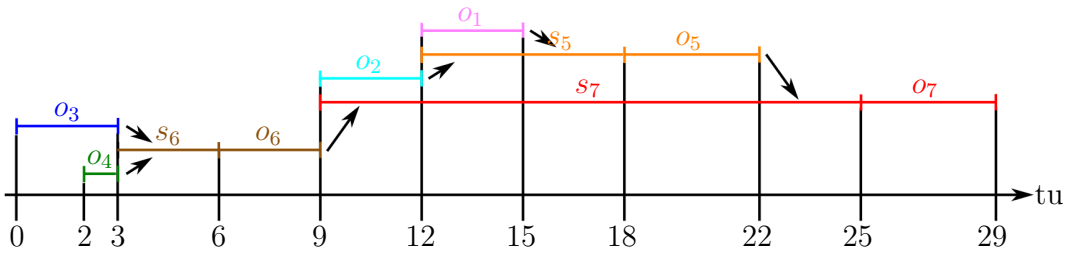

Figure 6.11: The scheduling result of case PCR in p1.[44]

14 and column 16, the results are much better, even with a small number of valves shown in $\#v$.

Compared with the conference version, three major improvements are proposed in the journal version:

1. Routing-convenient mapping with virtual boundary mentioned in Section 6.4.
2. Assurance of fluid paths to chip boundaries mentioned in Section 6.5.
3. Valve-actuation-aware routing applying rip-up and reroute method mentioned in Section 6.6.

which bring about better solutions in valve actuation as well as the sum of valves, and enhance the reliability of our method.

The routing-convenient mapping provides our model more flexibility in generating devices, while their connections are guaranteed with virtual boundaries and virtual areas of devices. For large designs that provide more options to the locations of devices, devices can be generated in best places when their locations are not strongly limited by the locations of their parent and child devices. Since the best locations for devices performing sequential operations may be far apart from each other, to route the possible long connections between them, our valve-actuation-aware routing method shows its benefit. In the journal version, we model corresponding valve actuations accurately for accessing every chamber in

the chip, and we further revise the results by rip-up and reroute. Compared with the routing in the conference version where we route fluid paths in the shortest length, in the journal version a longer path may be preferred if valve actuations led by routing this path can be reduced. As shown in Table 6.2, we achieve results with noticeable improvements for vs_{max} in column 8 and column 11 as well as for $\#v$ in column 9 and column 12. For example, compared with the conference version, for test case Exponential Dilution in policy 1, in the journal version we further reduce the largest number of valve actuations under both conservative and aggressive settings by more than 25%.

In the conference version, we only performed optimization under conservative setting, and the results under aggressive setting were directly derived from the results under conservative setting, where the program runtimes under aggressive setting approximate 0 and we thus omitted them in the table. In the journal version, we perform optimization for each test case both under conservative and aggressive setting.

In general, the program runtime denoted as T in Table 6.2 is not very stable, which strongly depends on the heuristics the optimization solver selects. However, a trend can still be observed that the program runtime typically exponentially increases with the problem size, which is mainly caused by the ILP modeling method that we apply. Though the scalibility is usually considered as an issue for ILP modeling method, it is not a serious problem for this work, since the matrix size of the general purpose architecture cannot be unlimitedly enlarged.

To show the working principles of our method intuitively, we take the synthesis result of case PCR with 7 mixing operations in policy 1 as an example. The input of our method is the scheduling result of this case with 3 time-units (tu) as the transport delay. We show the scheduling result in Figure 6.11 and the

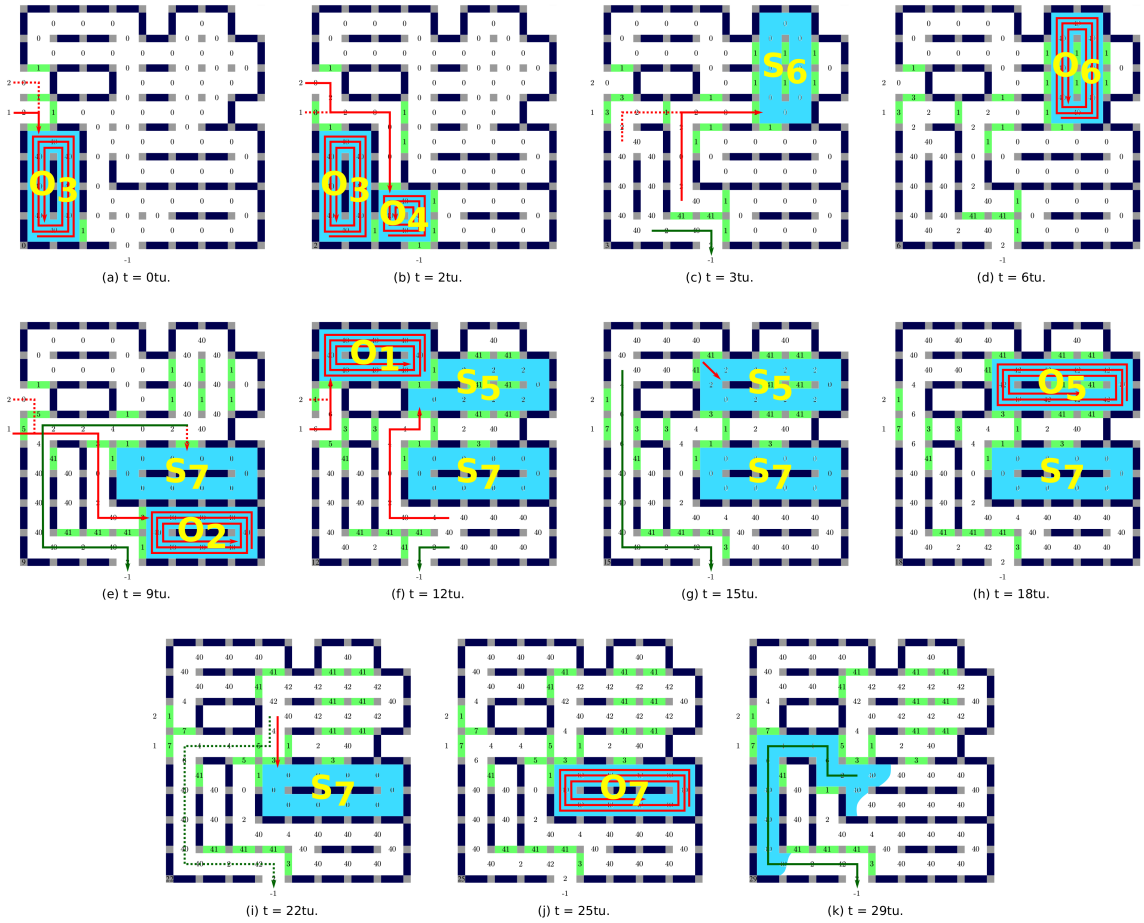

Figure 6.12: Snapshots of the synthesis result of test case PCR in policy 1 under conservative setting.[44]

synthesis result in Figure 6.12, in which o_1 and o_2 are the parent operations of o_5, o_3 and o_4 are the parent operations of o_6, and o_5, o_6 are the parent operations of o_7.

In Figure 6.12, closed valves are drawn in light color. The valves that are never actuated are removed and the area is left empty, just like the two valves at the top-right corner of the chip. In addition, if valves are only actuated once and no fluid flows through them while they are open, they are removed as well

and we build functionless walls drawn in dark color at the areas those valves once occupied. The numbers of valve actuations at every time moment are directly labeled on the corresponding valves. Fluid paths are represented by lines and their directions are indicated by the arrows on the lines. If two or more paths come from or go to the same region, the paths routed earlier are drawn in dashed lines, and the paths routed later are drawn in solid lines. Since not all of the products would go to next devices but some of them also would go to waste sink, for each dynamic device finishes its job we route a fluid path for it to the waste port. Unlike the fluid paths from input ports, fluid paths to waste port are drawn in dark color.

At t = 0tu as shown in Figure 6.12(a), o_3 starts and takes sample and reagent as inputs from port 1 and 2. The input from port 2 comes first, and is followed by the input from port 1.

Since o_3 is a mixing operation with a volume as 8 units, it occupies a 2×4 area in the chip, in which 2 internal valves are closed as internal boundaries of the mixer, and the other 8 internal valves are actuated for 40 times as pump valves to produce a circulation flow.

At t = 2tu as shown in Figure 6.12(b), o_4 starts and the dynamic device mapped by it is located adjacent to the device mapped by o_3. These two devices share the same closed valves as their outer boundaries. The input of o_4 from port 1 comes first this time, and the valve connected to port 1 at the chip boundary is closed when routing the fluid path from port 2.

At t = 3tu, storage s_6 is constructed immediately after o_3 and o_4 are finished as shown in Figure 6.12(c), which stores the products of o_3 and o_4. Some products of o_3 and o_4 go to their next device s_6, but some of them also go to waste sink. Note that the storage mapped by s_6 is placed in a distance from o_3 and o_4. In the

method in the conference version, this situation can not happen since the child devices are forced to be placed adjacent to their parent devices to prevent other devices from being inserted between them. With the concept of virtual boundaries proposed in the journal version, the virtual area of s_6 can be larger than the area it really occupies, and is adjacent to the mixers mapped by o_3 and o_4. Since the overlapping between virtual areas is prohibited, no device can be inserted between the devices mapped by o_3 and s_6 as well as o_4 and s_6, so that the connections between them can be guaranteed and directly constructed.

At t = 6tu s_6 changes into a device mapped by o_6 and starts to work as shown in Figure 6.12(d), which ends at t = 9tu as shown in Figure 6.12(e). Some of product of o_6 goes to s_7, and some other goes to the waste port. This fluid path to the waste port is not the shortest one in distance from the device mapped by o_6, but has the lowest cost according to our cost function (6.29). At the same time, o_2 starts after receiving inputs from port 1 and port 2.

At t = 12tu o_2 ends and sends its product to s_5 as well as to the waste port as shown in Figure 6.12(f). Also, o_1 starts while the device performing o_1 occupies a chamber of the storage for s_5 with the prerequisite that the remaining free area inside the storage is still enough to store its input from o_2. In our test case, according to the information from the sequencing graph of the bioassay, only 2 volume-unit product of o_5 comes from o_2 and 8 volume-unit product comes from o_1, thus the overlapping of 1 volume-unit chamber between the device mapped by o_1 and s_5 is allowed, since the storage mapped by s_5 only contains 2 volume-unit product from o_2 for the time being, and the feasibility check mentioned in Section 6.7 can pass.

In Figure 6.12(f), though storage s_5 overlaps with one of its parent device mapped by o_1, it is placed far away from its another parent device mapped by o_2,

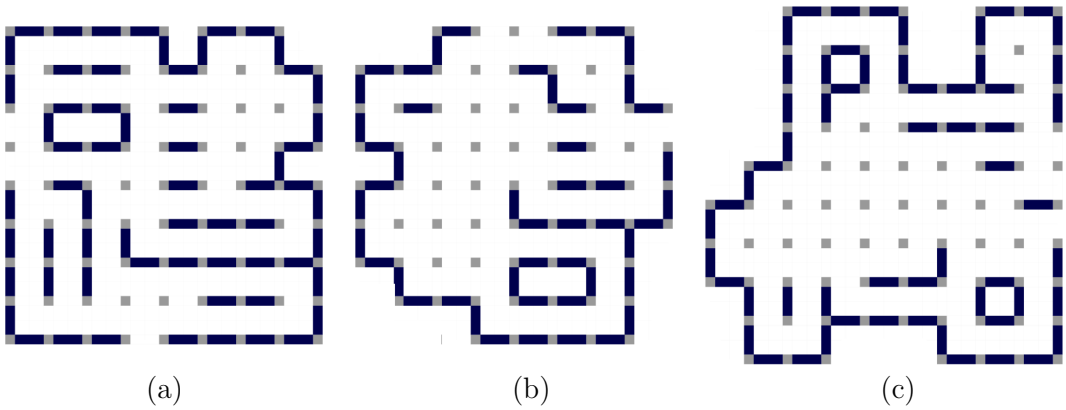

Figure 6.13: Different designs of PCR in different policies under conservative setting: (a) Policy 1. (b) Policy 2. (3) Policy 3.[44]

and their direct connection is blocked by the storage for s_7. In the journal version, this situation can happen since the device mapped by o_7 is the child device of the device mapped by o_5, thus the overlapping of their virtual areas is allowed when performing dynamic device mapping. However, when performing feasibility check for overlapping area among devices, we use the real boundaries of devices so that the check passes since the actual area of the device mapped by o_7 do not overlap with the actual area of the device mapped by o_5. Consequently, storage s_5 can be placed far from the device mapped by o_2, and storage s_7 can be placed in between them.

Though s_7 seems to obstruct the direct connection from the device mapped by o_2 to the device mapped by o_5, device like s_7 must be a storage and is not a blockage in most cases since one of its parent devices, e.g. the device mapped by o_5 in this case, just starts to work after receiving inputs from this direction connection. In this case, though the valve-actuation-aware routing method eventually routes this connection as a detour since this routing leads to the minimum valve actuation according to our cost function, a straight connection is still available,

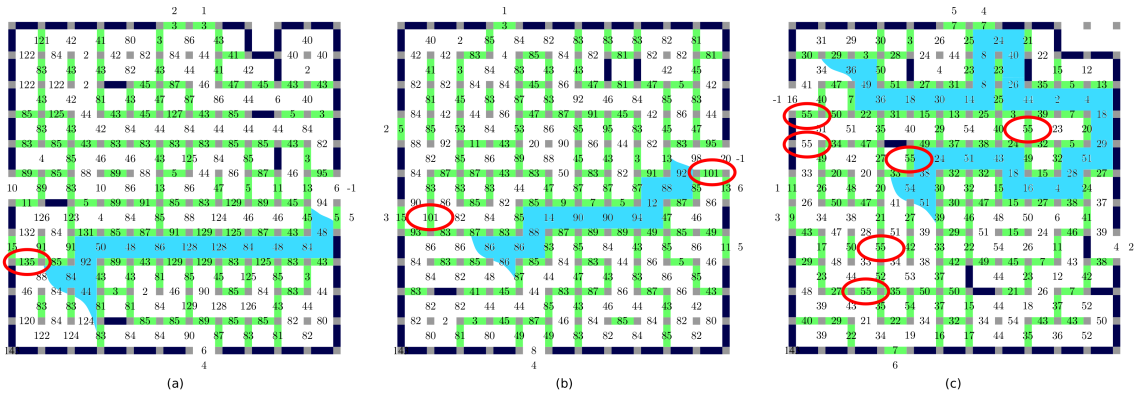

Figure 6.14: Different synthesis results of test case Exponential Dilution at t = 143tu by different methods: (a) With the method proposed in the conference version.[42] (b) With the method proposed in the journal version under conservative setting.[44] (c) With the method proposed in the journal version under aggressive setting.[44]

as s_7 only contains 2 volume-unit input from o_6 according to our test case and has 8 free volume units left that can be used for overlapping.

At t = 15tu as shown in Figure 6.12(g), o_1 ends and sends its product to s_5 and the waste port. Then o_5 starts to work at t = 18tu as shown in Figure 6.12(h). After o_5 ends at t = 22tu as shown in Figure 6.12(i), it sends its product to s_7 as well as to the waste port. Note that the fluid path from the device mapped by o_5 to the waste port has a similar shape with the paths from o_3 to s_6 in Figure 6.12(c), from o_3 as well as o_4 to the waste port in Figure 6.12(c), from o_6 to the waste port in Figure 6.12(e), and from o_1 to the waste port in Figure 6.12(g), because our rip-up and reroute method tend to minimize valve actuation regarding the current status of valves. Since these valves are used to form fluid paths or flow channels in devices at early time moments, they tend to be chosen to form flow channels for fluid paths as only few valve actuations are needed.

Finally, at t = 25tu as shown in Figure 6.12(j), the last operation o_7 starts, and it ends at t = 29tu as shown in Figure 6.12(k). We assume that for all test

cases, their final products leave the chip from the only output port, which is also regarded as the waste port. Note that the fluid path from the device mapped by o_7 to the waste port is routed in the same manner again as for other devices.

Figure 6.13 shows different designs of PCR in different policies under conservative setting, where designs in Figure 6.13(a) and Figure 6.13(b) are derived from a 8×8 valve matrix, and the design in Figure 6.13(c) is derived from a 9×9 valve matrix. These designs look very different from each other since compared with the sum of valves actually needed to implement the designs, the sum of virtual valves in these matrices are more than sufficient, and the synthesis results have shown some characteristics about the bioassays.

Figure 6.14 shows the performance of our method on relatively small virtual-valve matrices for test case Exponential Dilution by using different methods or under different settings, in which the status of valves and the sum of valve actuations show the final moment of the chip that it should be after finishing the whole bioassay at t = 143tu. Figure 6.14(a) shows the result of applying the method in the conference version. Since each time when a valve plays the role as pump valve, it needs to be actuated for 40 times under our setting. In Figure 6.14(a) there are several valves playing the role as pump valve for 3 times, while there are also several valves playing the role as pump valve for 0 times. For these valves, the difference in valve actuation can be larger than 120, and thus leads to a remarkable imbalance. This imbalance is significantly alleviated by our method proposed in the journal version. In Figure 6.14(b), the result is greatly improved as most valves play pump valve for 2 times, and the largest number of actuations decrease from 135 to 101.

Figure 6.14(c) shows the result of applying the method proposed in the journal version under aggressive setting, by which valve actuations are further

decreased. Note that in Figure 6.14(a), there is only one valve under the heaviest loading and needs to be actuated for 135 times. We circle out this certain valve in Figure 6.14(a). But in Figure 6.14(b) there are 2 valves actuated most for 101 times, and in Figure 6.14(c) there are 6 valves actuated for 55 times under the heaviest loading. This means that compared with the conference version, valve actuations in the journal version are not only decreased but also distributed evenly.

Figure 6.14(c) also demonstrates that a fluid path with the minimum valve actuations may not be the shortest one. In Figure 6.14(c), a long detour is chosen for forming the fluid path to transport the final product of a detector to the output port. Though this path is long, it takes advantage of existing flow channels and thus contributes to a better solution.

As a conclusion, in this chapter I have addressed a reliability problem of flow-based biochips due to unbalanced valve actuations. The problem is solved by the proposed reliability-aware synthesis including two steps as dynamic device mapping and fluid path routing based on a virtual valve-centered architecture with valve-role-changing concept. I first published my proposed method as a conference paper and then enhanced and published it as a journal paper. Compared with the conference version, I have revised the routing-convenient device mapping, assured fluid paths to chip boundaries, and proposed a valve-actuation-aware routing for fluid paths. Experimental results show the effectiveness of the proposed method.

7. Columba: Co-Layout Synthesis for Continuous-Flow Microfluidic Biochips

The conventional automated physical design approach for classical continuous-flow microfluidics is to separate the design process into flow-layer and control-layer design, which cannot handle the layer interactions properly, due to the lack of a global view. As a result, before 2016, there had been no physical design tool that could generate the complete layout of a chip including placement and routing solutions of both control and flow layers.

In 2016, my team proposed the first co-layout synthesis tool Columba that filled this design automation gap[47]. Columba took plain-text descriptions as inputs, and synthesized the layout of both control and flow layers simultaneously. The output of Columba was AutoCAD-compatible designs, which could be used for mask fabrication.

In order to model layer interactions in a systematic manner, we proposed the first physical-design module models for important microfluidic devices and components including mixer, reaction chamber, switch and port. The inner structures

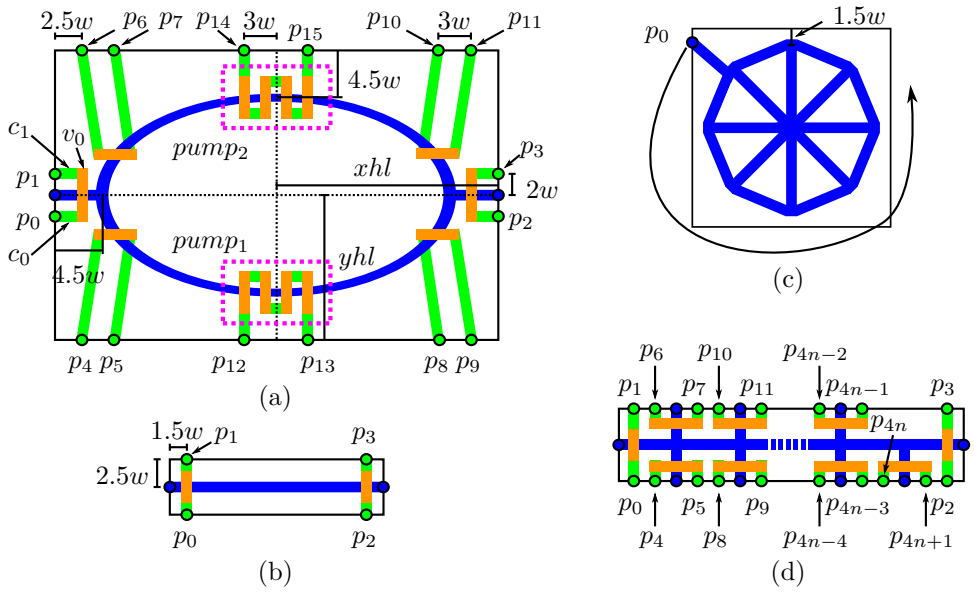

Figure 7.1: Physical-design module models for: (a) Mixer. (b) Reaction Chamber. (c) In-/Outlet (fluid). (d) Switch.[47]

of the proposed models are shown in Figure 7.1, where flow channels are indicated by blue lines, control channels by green lines, and valves by orange rectangles. We allowed modification of our module models to meet various design requirements, including the change of component dimension, channel width, minimum spacing distance, etc. We also provided alternative pin-locations to enhance the flexibility of our design. For example, in a mixer module as proposed in Figure 7.1(a), we provided two potential locations of peristalsis pumps: *pump1* and *pump2*, the selection of which would be made by Columba during the synthesis process. Once the location and orientation of our modules are determined, the location of valves are determined automatically. As layer interactions only happen at valves, our module models serve as the basic units for both control and flow layer design.

Columba performed a progressive integer-linear-programming (P-ILP) based modeling method for layout generation, which consisted of four phases: 1. global

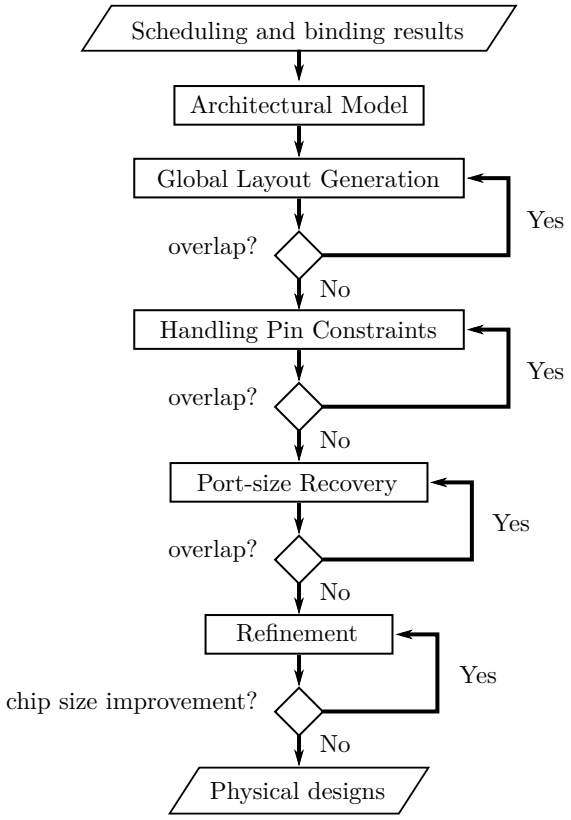

Figure 7.2: The flowchart of Columba.[47]

layout generation; 2. valve-level restoration; 3. in-/outlet restoration; and 4. iterative refinement. A flowchart of the proposed method is shown in Figure 7.2.

In the following, we describe our proposed method in detail. Similar descriptions can also be found in our published paper[47].

7.1 Global Layout Generation

7.1.1 Channel model

As the backbone of the global layout, we propose a directed channel model which consists of several bending points and channel segments. As shown in Fig-

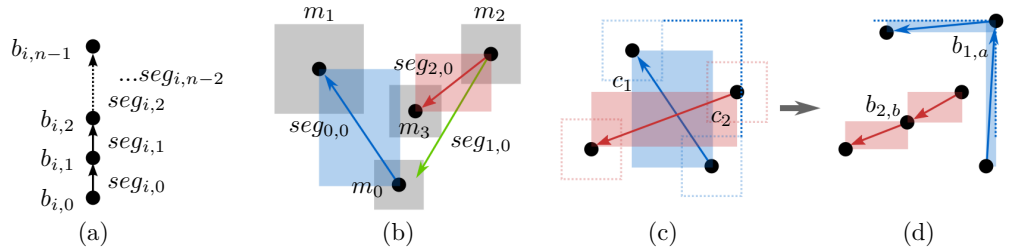

Figure 7.3: (a) Channel model. (b) An example of initial states of channels in global layout. (c) An example of channel crossing. (d) An example of adding new bending points.[47]

ure 7.3(a), the bending points of the i-th channel c_i are indexed as $b_{i,0}, \cdots, b_{i,n-1}$, and the channel segments are indexed as $seg_{i,0}, \cdots, seg_{i,n-2}$, thereof $b_{i,k}$ and $b_{i,k+1}$ are also called the terminals of $seg_{i,k}$, and $b_{i,0}$ and $b_{i,n-1}$ are also called the terminals of c_i. $n \in \mathbb{N}$ indicates the number of bending points.

We represent the location of the i'-th module containing valves by its center points $(m_{i',x}, m_{i',y})$ in a coordinate system, together with $m_{i',x} \pm \frac{1}{2}m_{i',w}$, $m_{i',y} \pm \frac{1}{2}m_{i',h}$ indicating the location of the module boundaries, $m_{i',w}$ and $m_{i',h}$ indicating the width and height of the i'-th module.

In phase 1, for the sake of model reduction, instead of connecting each channel with specific pins of modules, we connect channels with the center points of modules. The incoming (outgoing) control channels connecting the same pair of modules are regarded as a whole *channel bundle* sharing the same bending points. We index the j-th channel bundle as cb_j, along with a new variable ncb_j indicating the number of control channels merged by cb_j. We assign a group of control ports with no area cost (see Section 7.3) to each module containing valves. The number of ports which a port group eventually contains is determined by the number of control channels connected to it.

7.1.2 Pressure Source Sharing

Valves inside the same module are required to work simultaneously with different actuation patterns, and therefore need to be driven by individual pressure sources. However, valves in different modules that do not have any overlapping working period according to the scheduling results can share the same pressure sources to reduce control efforts.

Therefore, besides an incoming control channel connected to a pressure source, every valve v_a may also possess an outgoing control channel, which is connected with another valve v_b as the incoming control channel of v_b in a different module, thus v_a and v_b are connected in series to share the same pressure source.

We introduce the following constraints to describe this pressure sharing scenario for each module $m_{i'} \in M$:

$$\sum_{\forall cb_j \in C_{in,i'}} ncb_j = \mathcal{V}_{i'}, \tag{7.1}$$

$$\sum_{\forall cb_j \in C_{in,i'}} ncb_j \geq \sum_{\forall cb_j \in C_{out,i'}} ncb_j \tag{7.2}$$

where M is the set of all modules, $C_{in,i'}$ ($C_{out,i'}$) is the set of all incoming (outgoing) channel bundles of $m_{i'}$, and $\mathcal{V}_{i'}$ is a constant representing the number of valves in $m_{i'}$. (7.1) means that the number of incoming control channels connected to $m_{i'}$ must equal the number of valves in $m_{i'}$, since every valve possesses an incoming control channel as its pressure source. (7.2) means that the number of the outgoing control channels connected to $m_{i'}$ should be no more than the number of the incoming control channels, since not all valves necessarily possess

outgoing control channels.

7.1.3 Channel Non-crossing Limitations

As shown in Figure 7.3(b), each channel possesses in its initial state only one segment formed by two terminals, which are the center points of the two modules connected by this channel. In order to avoid the crossing among channels in the same layer, we view each segment as the diagonal of a rectangle with exclusive chip area, and prohibit overlapping among these areas: suppose that $b_{i,k}, b_{i,k+1}$ and $b_{j,l}, b_{j,l+1}$ are the respective terminals of two different segments $s_{i,k}$ and $s_{j,l}$, and the coordinate of a bending point (terminal) is represented as $(b_{\cdot,\cdot,x}, b_{\cdot,\cdot,y})$. There are four types of possible locations allowed for these terminals to avoid area overlapping, which can be transformed into (7.3)(7.4)(7.5)(7.6):

$$\begin{aligned} b_{i,k,x} &\le b_{j,l,x} + q_{le}\Phi - w, \quad b_{i,k,x} \le b_{j,l+1,x} + q_{le}\Phi - w, \\ b_{i,k+1,x} &\le b_{j,l,x} + q_{le}\Phi - w, \quad b_{i,k+1,x} \le b_{j,l+1,x} + q_{le}\Phi - w, \end{aligned} \tag{7.3}$$

$$\begin{aligned} b_{i,k,x} &\ge b_{j,l,x} + q_{ri}\Phi - w, \quad b_{i,k,x} \ge b_{j,l+1,x} + q_{ri}\Phi - w, \\ b_{i,k+1,x} &\ge b_{j,l,x} + q_{ri}\Phi - w, \quad b_{i,k+1,x} \ge b_{j,l+1,x} + q_{ri}\Phi - w, \end{aligned} \tag{7.4}$$

$$\begin{aligned} b_{i,k,y} &\le b_{j,l,y} + q_{bo}\Phi - w, \quad b_{i,k,y} \le b_{j,l+1,y} + q_{bo}\Phi - w, \\ b_{i,k+1,y} &\le b_{j,l,y} + q_{bo}\Phi - w, \quad b_{i,k+1,y} \le b_{j,l+1,y} + q_{bo}\Phi - w, \end{aligned} \tag{7.5}$$

$$\begin{aligned} b_{i,k,y} &\ge b_{j,l,y} + q_{up}\Phi - w, \quad b_{i,k,y} \ge b_{j,l+1,y} + q_{up}\Phi - w, \\ b_{i,k+1,y} &\ge b_{j,l,y} + q_{up}\Phi - w, \quad b_{i,k+1,y} \ge b_{j,l+1,y} + q_{up}\Phi - w, \end{aligned} \tag{7.6}$$

$$q_{ri} + q_{le} + q_{up} + q_{bo} = 3 + q_p \tag{7.7}$$

where w is the minimum spacing distance between two segments, which is larger than the value specified in the design check rules (see Section 7.3) and Φ is an

extremely large auxiliary integer variable. q_{le}, q_{ri}, q_{bo}, q_{up}, and q_p are auxiliary binary variables, thereof q_p is the *Lagrange multiplier*, the minimization of which is included in the optimization targets. If $q_p = 0$, one of (7.3)(7.4)(7.5)(7.6) has to be non-trivial according to (7.7), and a feasible solution of this inequality group indicates locations of two segments without area overlap.

If the overlapping of two segments is inevitable with the current bending points status, q_p has to be set to 1. In this situation, phase 1 will be rerun and new bending points will be added to the channels with conflicts, thus enabling a detour around the conflict area. As shown in Figure 7.3(c), the initial states of channel c_1 and c_2 result in an inevitable crossing. Therefore, new bending points $b_{1,a}$ and $b_{2,b}$ are added to c_1 and c_2 respectively. The locations of the existing bending points can be adjusted within a limited floating range according to their previous locations, and the coordinates of $b_{1,a}$ and $b_{2,b}$ are constrained by the locations of these existing binding points as shown in Figure 7.3(c). With $b_{1,a}$ and $b_{2,b}$, now the crossing can be avoided as shown in Figure 7.3(d).

7.1.4 Module Placement

The non-overlapping limitations of modules are realized in a similar manner as Section 7.1.3:

$$m_{i',x} - \frac{1}{2}m_{i',w} \geq m_{j',x} + \frac{1}{2}m_{j',w} - q_{ri}\Phi, \tag{7.8}$$

$$m_{i',x} + \frac{1}{2}m_{i',w} \leq m_{j',x} - \frac{1}{2}m_{j',w} + q_{le}\Phi, \tag{7.9}$$

$$m_{i',y} - \frac{1}{2}m_{i',h} \geq m_{j',y} + \frac{1}{2}m_{j',h} - q_{up}\Phi, \tag{7.10}$$

$$m_{i',y} + \frac{1}{2}m_{i',h} \leq m_{j',y} - \frac{1}{2}m_{j',h} + q_{bo}\Phi, \tag{7.11}$$

$$q_{ri} + q_{le} + q_{up} + q_{bo} = 3 + q_p. \tag{7.12}$$

If q_{ri} (q_{le}, q_{up}, q_{bo}) is set to 0, $m_{i'}$ will be placed to the $right$ ($left$, $upper$, $bottom$) side of $m_{j'}$.

In our module models, the flow channels are connected to a mixer module or a reaction chamber module from opposite directions. For the convenience of channel routing, we rule that if m_b and m_c are two modules connected to module m_a, then m_b and m_c must be placed at opposite directions relative to m_a. For example, if m_b is placed to the right side of m_a, then m_c has to be placed to the left side of m_a. We specify this rule by modifying the auxiliary binary variables in (7.8)-(7.12).

7.1.5 Objective

We represent the side lengths of a chip as two continuous variables s_x and s_y, and the set of all bending points as B. Then we introduce the following constraints to

all $b_{i,k} \in B$:

$$b_{i,k,x} \geq d \quad \wedge \quad b_{i,k,x} \leq s_x - d \quad \wedge \quad b_{i,k,y} \geq d \quad \wedge \quad b_{i,k,y} \leq s_y - d \tag{7.13}$$

thereof d is the minimum spacing distance from the chip boundaries to a bending point. If $b_{i,k}$ is the center point of a module $m_{i'}$, we increase d by $\frac{1}{2}max\{m_{i',w}, m_{i',h}\}$. Since the chip area is a quadratic variable, we introduce a new continuous variable $s_d = max\{s_x, s_y\}$ to represent the dimension of the chip.

Therefore, we formulate our minimization objective as:

$$\alpha s_d + \alpha_1 s_x + \alpha_2 s_y + \beta \sum_{\forall cb_i \in C} cb_{i,l} + \gamma \sum_{\forall cb_i \in C_P} ncb_i + \sigma \sum_{\forall q_p \in E} q_p \tag{7.14}$$

thereof C is the set of all channel bundles, C_p is the set of all channel bundles that are connected with port groups, E is the set of Lagrange multipliers, $cb_{i,l}$ is the length of cb_i, and α (together with α_1 and α_2), β, γ, σ are thus the adjustable weight coefficients of chip area, total channel bundle lengths, the number of ports and Lagrange multipliers respectively, thereof $\alpha_1, \alpha_2 \ll \alpha$ are used to prevent the abuse of chip area, and α is increased progressively in the following phases.

7.2 Handling Pin Constraints

With the modelling results from phase 1 indicating the module locations and channel connections, we can specify the locations of valves and pins inside a module, which enables us to split a channel bundle into real channels connected with the pins for an accurate layout as shown in Figure 7.4(a), instead of with the center point of a module.

For a given module $m_{i'}$, we represent the set of all incoming channels con-

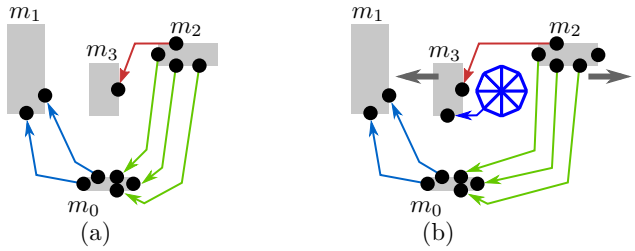

Figure 7.4: (a) An example of accurate layout. (b) An example of port-size recovery.[47]

nected to $m_{i'}$ as C_{in}, the set of all outgoing channels connected to $m_{i'}$ as C_{out}, and the set of pins in $m_{i'}$ as P, thereof all pins are indexed according to the module models shown in Figure 7.1, and the location of a pin $p_j \in P$ is represented as $(m_{i',x} + \kappa_{j,x}, m_{i',y} + \kappa_{j,y})$, where $\kappa_{j,x}$ and $\kappa_{j,y}$ are constants indicating the distance between p_j and the center point of $m_{i'}$.

Therefore, we can introduce the following constraints to locate the terminals $b_{i,k}$ of control channels $c_i \in C_{in} \cup C_{out}$ connected to $m_{i'}$:

$$b_{i,k,x} = m_{i',x} + \sum_{0 \le j < |P|} q_{i,j} \kappa_{j,x}, \tag{7.15}$$

$$b_{i,k,y} = m_{i',y} + \sum_{0 \le j < |P|} q_{i,j} \kappa_{j,y}, \tag{7.16}$$

thereof $q_{i,j}$ is an auxiliary binary variable representing the selection of pins: if $q_{i,j}$ is set to 1, channel c_i will be connected to pin p_j.

Then we introduce the following constraints to ensure that c_i will be connected to exactly one pin of $m_{i'}$ (7.17), and a pin p_j can be connected with at

most one channel (7.18):

$$\forall c_i \in C_{in} \cup C_{out}, \sum_{0 \le j < |P|} q_{i,j} = 1, \tag{7.17}$$

$$0 \le j \le |P|, \sum_{0 \le i < |C_{in} \cup C_{out}|} q_{i,j} \le 1. \tag{7.18}$$

As mentioned in Figure 7.1 and Section 7.1.2, each valve in our modules possesses two pins, which can be indexed as $p_{j'}$ and $p_{j'+1}$, $j' \in \{2k | k \in \mathbb{N}_0\}$.

For a valve controlling fluid flow, one of its pins must be connected with an incoming control channel, while the other pin can be connected with an outgoing control channel or be left unused. Therefore, the number of pins connected with outgoing control channels must be no more than the number of pins connected with incoming control channels. This is described by the following constraints:

$$\sum_{i \in \{i | c_i \in C_{in}\}} q_{i,j'} \ge \sum_{\check{i} \in \{\check{i} | c_{\check{i}} \in C_{out}\}} q_{\check{i},j'+1}, \tag{7.19}$$

$$\sum_{i \in \{i | c_i \in C_{in}\}} q_{i,j'+1} \ge \sum_{\check{i} \in \{\check{i} | c_{\check{i}} \in C_{out}\}} q_{\check{i},j'}. \tag{7.20}$$

Since we have two possible locations for a pump in our mixer module, not all valves forming peristalsis pumps need to be connected with control channels. As shown in Figure 7.1(a), if we regard $pump_1$ and $pump_2$ as valves possessing two pins, only one of them will be connected with a control channel and the other will be left unused. Therefore, we introduce the following constraints to describe

channel connections for mixer modules:

$$\sum_{i \in \{i | c_i \in C_{in}\}} (q_{i,j'} + q_{i,j'+1}) \leq 1, \tag{7.21}$$

$$\sum_{i \in \{i | c_i \in C_{in}\}, j \in \{12,13,14,15\}} q_{i,j} = 1. \tag{7.22}$$

We also put the non-crossing constraints mentioned in Section 7.1.3 on the new control channels generated in this phase, and rerun the optimization until no channel crossing exists.

7.3 Port Module Restoration

Since a port module contains only one pin, which can be located anywhere on the module boundary, the connection between ports and other modules is not so complicated as the connection between two modules containing valves. Therefore, for the sake of model reduction, the area costs of ports are ignored in phase 1 and phase 2.

In this phase, we restore the ports as modules with given side length. Since we have kept a relatively large minimum spacing distance between channels and modules in previous phases as mentioned in Section 7.1.3 and Section 7.1.4, it is not difficult to find suitable locations for these ports with little draw-and-push efforts as shown in Figure 7.4(b). Then we put the non-crossing constraints on the channels connected with ports and other modules, and rerun the optimization until no channel crossing exists.

Table 7.1: Generated design features.[47]

	$D(mm^2)$	$L(mm)$	$\#(m,\ r,\ s,\ fp,\ cp)$	T
kinase act.[6]	15.05 × 15.05	163.54	2, 2, 3, 7, 24	5m22s
acid proc.[11]	18.35 × 18.15	252.83	3, 3, 3, 11, 40	9m5s
ChIP (4IP)[51]	27.95 × 26.65	298.25	5, 4, 2, 17, 44	9m56s
mRNA iso.[18]	22.77 × 24.30	564.01	4, 4, 3, 18, 54	34m42s
ChIP (10IP)	38.15 × 38.11	556.42	11, 10, 2, 23, 100	46m10s

D: chip dimension. L: total length of channels. $\#m$, $\#r$, $\#s$, $\#fp$, $\#cp$: number of mixers, reaction chambers, switches, fluid ports, and control ports. T: program runtime.

7.4 Refinement

In the last phase of our modelling process, we replace the above mentioned large minimum spacing distance with the actual requirement according to[30], and re-run the optimization for better utilization of the chip area.

7.5 Experimental results

We implemented our method in C++ on a computer with 2.40 GHz CPU. The proposed model is linear and thus the optimization problem can be solved by the Mixed Integer Linear Programming (MILP) solver Gurobi[10].

Table 7.1 shows the feature values of the automatically-generated designs by Columba. The first four test cases are from[6,11,18,51] and are listed with the order as their sizes. To show the scalability of Columba, we design a 10 Immunoprecipitation (IP) chip (ChIP (10IP)) by duplicating 6 more parallel IP process as the fifth test case. The number of the synthesized control ports $\#cp$ as well as the program runtime T increase smoothly proportional to the design complexity, which is denoted by $\#m$ (mixers), $\#r$ (reaction chambers), $\#s$ (switches), $\#fp$

(fluid ports) as the given inputs from scheduling results.

Figure 7.5 shows the comparison between the manual design and the automatic design by Columba for ChIP (4IP). Blue and green lines in the figure indicate the flow and control channels, small blue and green circles represent the fluid and control ports, big blue ellipses represent the mixers, pink rectangles represent the reaction chambers, and orange segments represent the valves. Since the dimension of each mixer is clearly specified as $7.5mm \times 3.8mm$ in[51], we can estimate the chip size of the manual design, which is larger than the automatic design generated by Columba.

Figure 7.6 shows the temporary resultant graphs of test case ChIP(10IP). The global layout is determined in phase 1 as shown in Figure 7.6(a); in phase 2, channel bundles are split into real channels and connected to pins of modules instead of the central points of modules as shown in Figure 7.6(b); and in phase 3, chip ports are restored as modules with area cost as shown in Figure 7.6(c). Since the weight coefficient α of area cost is increased progressively, the chip area is gradually reduced. Figure 7.7 shows the snapshot of the final design generated by Columba for ChIP(10IP), which, though large and complicated, requires less than 1 hour for the entire synthesis.

One of the outputs of Columba is an AutoCAD script file. The script can be run to generate an AutoCAD design automatically as shown in Figure 7.8. As currently AutoCAD is the most popular design tool for biochip designers, minor adjustments via AutoCAD can be easily made for these designers if they have some new ideas even after the design is already generated by Columba. In the meanwhile, this function also demonstrates that Columba can be easily and seamlessly integrated into the current design-and-manufacturing flow, lowering the barrier for the designers to use Columba.

Columba is the first tool that can automatically transform a plain-text netlist description into a valid microfluidic chip design with all necessary features. Compared with the manual design, applying Columba not only can save the design effort but the generated designs are guaranteed to be fault-free, since all design rules can be modeled as mathematical constraints in Columba and any constraint violation cannot escape from the check by the computer.

Columba aims to be the first commercial design automation tool for the mainstream continuous-flow microfluidics. Future work of Columba includes: (1) developing user interface; (2) extending the library of module models.

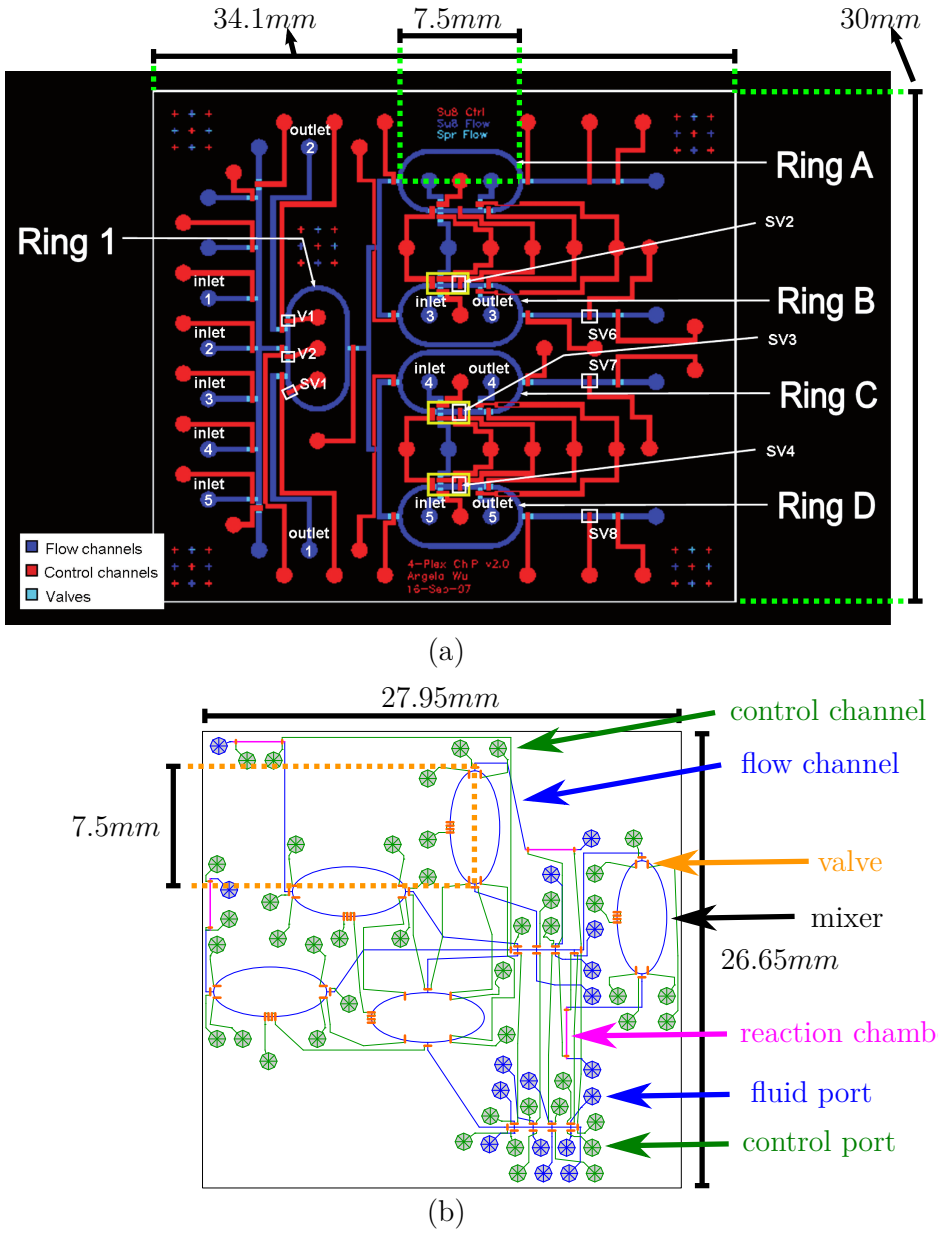

Figure 7.5: ChIP[51] design: (a) Manual (b) Columba.[47]

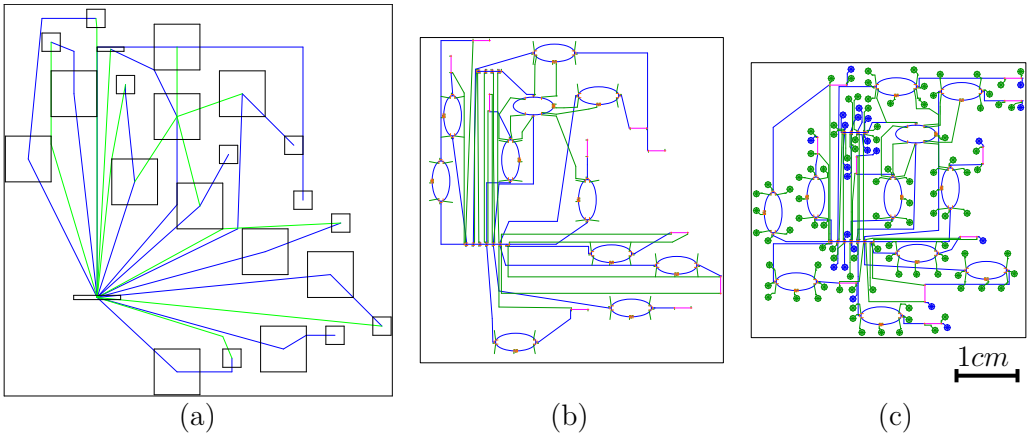

Figure 7.6: Temporary resultant graphs of test case ChIP(10IP): (a) Phase 1. (b) Phase 2. (c) Phase 3.[47]

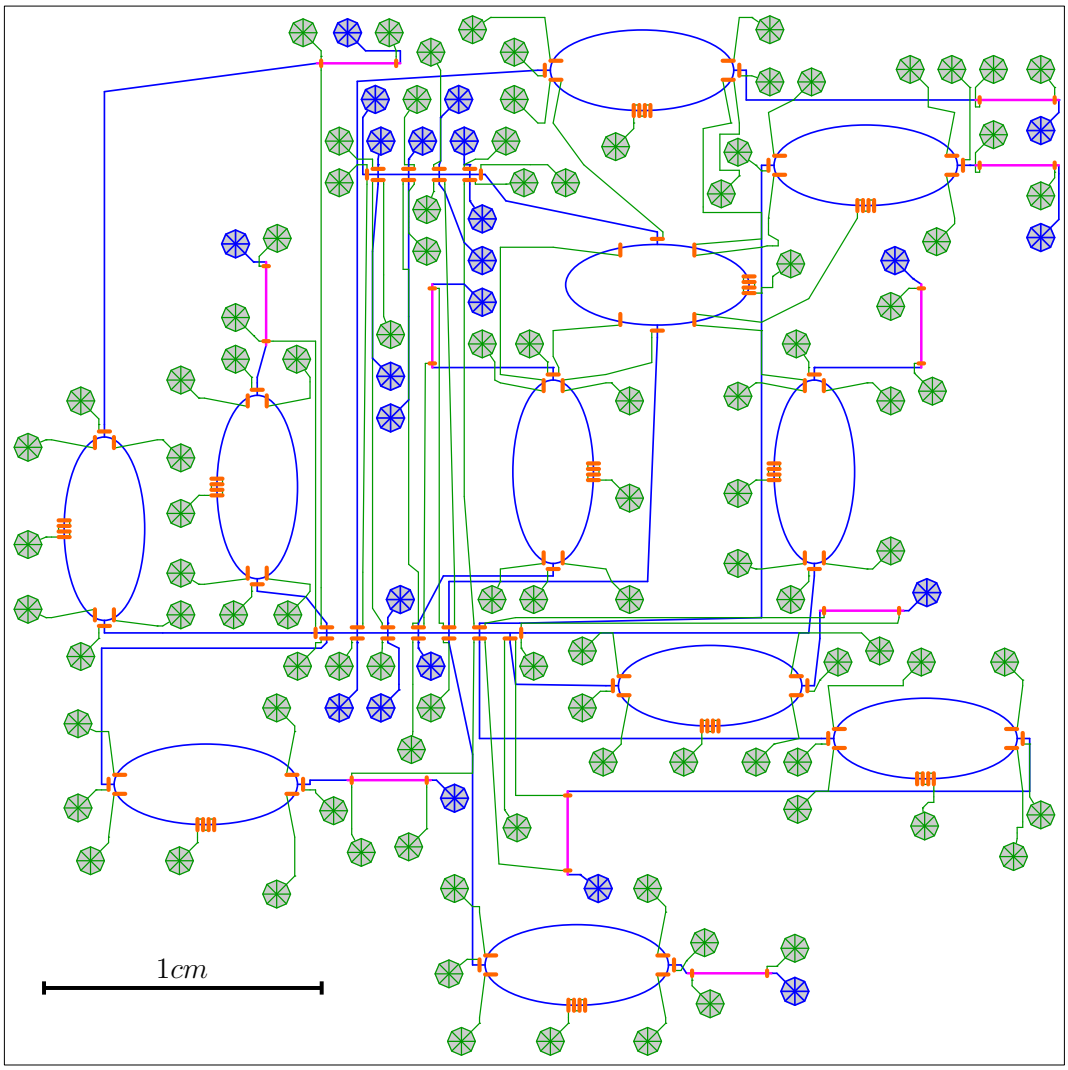

Figure 7.7: The final design of test case ChIP(10IP).[47]

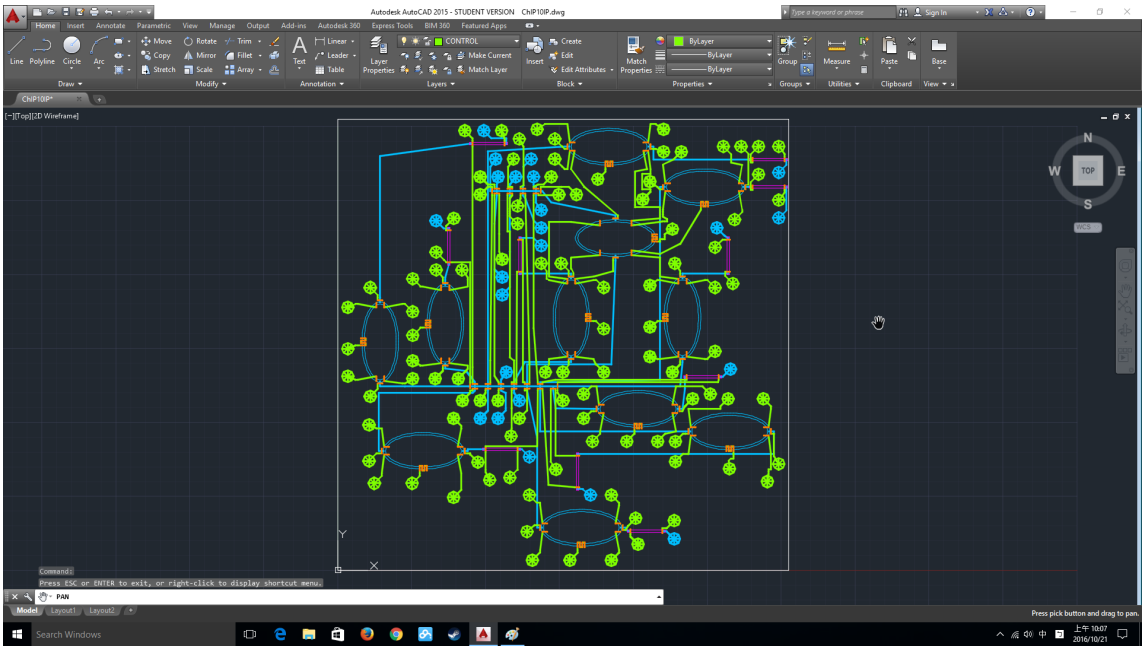

Figure 7.8: The ChIP(10IP) final design exported by AutoCAD and automatically generated by Columba.

8. Conclusion

With the endeavor over one decade, design automation research for continuous-flow microfluidics is approaching the state-of-the-art technology in bioengineering. In the front-end field, device/component library is continuously extended, and mature high-level synthesis algorithms have been developed. In the back-end field, the characteristics of representative microfluidic devices/components, the interaction among control channel routing, flow channel routing, and device/component placement, have been well studied. However, there remain two main challenges in design automation research for continuous-flow microfluidics.

The first challenge is the fact that currently there is no standard in microfluidic manufacturing. So far, most design automation research is based on the multi-layered valve technology developed by Quake's group at Stanford University. Though Quake's group undoubtedly holds a leading position in bioengineering, there are many more bioengineers working on different technologies, most of which are developed on their own. Different technologies lead to different design rules, and in some new technologies there are not even any explicit design rules. Therefore, to develop a design tool that everyone can be satisfied with is very difficult.

Another challenge is presented by large-scale-integrated designs. Though

both bioengineers and design automation engineers claim that large-scale-integration is the trend for future microfluidics, so far there has been no mature large-scale-integrated design, except the designs that have very homogeneous architecture. Even when a design automation tool can generate a large-scale-integrated design, there is no corresponding real assay that can be performed on it, and no comparison can be made between the manual design and the automatically-generated large-scale-integrated design.

To conquer these challenges, a close cooperation between bioengineers and design automation engineers is required. This, however, is difficult, since currently most bioengineers tend to invent small-scaled devices only to demonstrate the usability of their own technology, and neither standardization nor large-scale-integration brings them direct benefits. Nevertheless, both standardization and large-scale-integration are necessary conditions for microfluidics to become the dominant platform in performing bioassays, which needs continuous effort from both the bioengineering and design automation field.

Bibliography

[1] Amin, A. M., Thottethodi, M., Vijaykumar, T., Werely, S., & Jacobson, S. C. (2007). Aquacore: a programmable architecture for microfluidics. In *Proc. Int. Symp. on Comput. Archi.* (pp. 254–265).

[2] Amin, N., Thies, W., & Amarasinghe, S. P. (2009). Computer-aided design for microfluidic chips based on multilayer soft lithography. In *Proc. Int. Conf. Comput. Des.* (pp. 2–9).

[3] Chakrabarty, K. & Su, F. (2006). *Digital Microfluidic Biochips: Synthesis, Testing, and Reconfiguration Techniques*. Boca Raton, FL: CRC Press.

[4] Cho, C.-H., Cho, W., Ahn, Y., & Hwang, S.-Y. (2007). PDMS–glass serpentine microchannel chip for time domain PCR with bubble suppression in sample injection. *J. Micromech. Microeng.*, 17(9), 1810–1817.

[5] de la Guardia, M. & Garrigues, S. (2011). *Challenges in Green Analytical Chemistry*. Royal Society of Chemistry.

[6] Fang, C., Wang, Y., Vu, N. T., Lin, W.-Y., Hsieh, Y.-T., Rubbi, L., Phelps, M. E., Müschen, M., Kim, Y.-M., Chatziioannou, A. F., Tseng, H.-R., & Graeber, T. G. (2010). Integrated microfluidic and imaging platform for a

kinase activity radioassay to analyze minute patient cancer samples. *Cancer Res.*, 70(21), 8299–8308.

[7] Fidalgo, L. M. & Maerkl, S. J. (2011). A software-programmable microfluidic device for automated biology. *Lab on a Chip*, 11, 1612–1619.

[8] Fluidigm Corporation (2016). *C1 system.* https://www.fluidigm.com/products/c1-system.

[9] Gomez-Sjoeberg, R., Leyrat, A. A., Pirone, D. M., Chen, C. S., & Quake, S. R. (2007). Versatile, fully automated, microfluidic cell culture system. *Anal. Chem*, 79, 8557–8563.

[10] Gurobi Optimization, I. (2015). Gurobi optimizer reference manual.

[11] Hong, J. W., Studer, V., Hang, G., Anderson, W. F., & Quake, S. R. (2004). A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotechnology*, 22(4), 435–439.

[12] Hu, K., Yu, F., Ho, T.-Y., & Chakrabarty, K. (2014). Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 33(10), 1463–1475.

[13] Jiang, X., Shao, N., Jing, W., Tao, S., Liu, S., & Sui, G. (2014). Microfluidic chip integrating high throughput continuous-flow PCR and DNA hybridization for bacteria analysis. *Talanta*, (pp. 246–250).

[14] Li, M., Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2016). Sieve-valve-aware synthesis of flow-based microfluidic biochips considering specific

biological execution limitations. In *Proc. Design, Automation, and Test Europe Conf.* (pp. 624–629).

[15] Lin, C.-X., Liu, C.-H., Chen, I.-C., Lee, D. T., & Ho, T.-Y. (2014). An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips. In *Proc. Design Autom. Conf.* (pp. 141:1–141:6).

[16] Liu, J., Enzelberger, M., & Quake, S. (2002). A nanoliter rotary device for polymerase chain reaction. *Electrophoresis*, 23, 1531–1536.

[17] Luby, M. (1986). A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4), 1036–1053.

[18] Marcus, J. S., Anderson, W. F., & Quake, S. R. (2006). Microfluidic single-cell mRNA isolation and analysis. *Anal. Chem.*, 78, 3084–3089.

[19] Marcy, Y., Ishoey, T., Lasken, R. S., Stockwell, T. B., Walenz, B. P., Halpern, A. L., Beeson, K. Y., Goldberg, S. M. D., & Quake, S. R. (2007a). Nanoliter reactors improve multiple displacement amplification of genomes from single cells. *PLoS Genet*, 9(3), e155.

[20] Marcy, Y., Ouverney, C., Bik, E. M., Lösekann, T., Ivanova, N., Martin, H. G., Szeto, E., Platt, D., Hugenholtz, P., Relman, D. A., & Quake, S. R. (2007b). Dissecting biological "dark matter" with single-cell genetic analysis of rare and uncultivated tm7 microbes from the human mouth. *Proc. Nat. Acad. Sci.*, 104, 11889–11894.

[21] Mark, D., Haeberle, S., Roth, G., von Stetten, F., & Zengerle, R. (2010). Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chem. Soc. Rev.*, 39, 1153–1182.

[22] Micheli, G. D. (1994). *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education.

[23] Minhass, W. H., Pop, P., & Madsen, J. (2011). System-level modeling and synthesis of flow-based microfluidic biochips. In *Proc. Int. Conf. Compil., Arch. and Syn. Embed. Sys.* (pp. 225–234).

[24] Minhass, W. H., Pop, P., Madsen, J., & Blaga, F. S. (2012). Architectural synthesis of flow-based microfluidic large-scale integration biochips. In *Proc. Int. Conf. Compil., Arch. and Syn. Embed. Sys.* (pp. 181–190).

[25] Minhass, W. H., Pop, P., Madsen, J., & Ho, T.-Y. (2013). Control synthesis for the flow-based microfluidic large-scale integration biochips. In *Proc. Asia and South Pacific Des. Autom. Conf.* (pp. 205–212).

[26] National Human Genome Research Institute (2016). *Cost per Genome*. http://www.genome.gov/sequencingcosts/.

[27] Osoegawa, K., Mammoser, A. G., Wu, C., Frengen, E., Zeng, C., Catanese, J. J., & de Jong, P. J. (2001). A bacterial artificial chromosome library for sequencing the complete human genome. *Genome Research*, 11, 483–496.

[28] Oxford Genomics Centre (2015). *Single-cell genomics*. http://www.well.ox.ac.uk/ogc/single-cell-genomics.

[29] Ren, H., Srinivasan, V., & Fair, R. (2003). Design and testing of an interpolating mixing architecture for electrowetting-based droplet-on-chip chemical dilution. In *Int. Conf. on Solid-State Sensors, Actuators and Microsystems* (pp. 619–622).

[30] Stanford Foundry (2016). *Basic Design Rules.* http://web.stanford.edu/group/foundry.

[31] Su, F. & Chakrabarty, K. (2004). Architectural-level synthesis of digital microfluidics-based biochips. In *Proc. Int. Conf. Comput.-Aided Des.* (pp. 223–228).

[32] Su, F. & Chakrabarty, K. (2005). Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. In *Proc. Design Autom. Conf.* (pp. 825–830).

[33] Su, F. & Chakrabarty, K. (2008). High-level synthesis of digital microfluidic biochips. *ACM J. Emerg. Technol. Comput. Syst.*, 3(4).

[34] Su, F., Chakrabarty, K., & Fair, R. B. (2006). Microfluidics-based biochips: Technology issues, implementation platforms, and design-automation challenges. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 25(2), 211–223.

[35] Thorsen, T., Maerkl, S. J., & Quake, S. R. (2002). Microfluidic large-scale integration. *Science*, 298(5593), 580–584.

[36] Tripp, S. & Grueber, M. (2011). *Economic impact of the Human Genome Project.* Battelle.

[37] Tseng, K.-H., You, S.-C., Liou, J.-Y., & Ho, T.-Y. (2013a). A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization. In *Proc. Int. Symp. Phy. Des.* (pp. 123–129).

[38] Tseng, T.-M., Chao, M. C. T., Lu, C.-P., & Lo, C.-H. (2009). Power-switch routing for coarse-grain MTCMOS technologies. In *Proc. Int. Conf. Comput.-Aided Des.* (pp. 39–46).

[39] Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2013b). Post-route alleviation of dense meander segments in high-performance printed circuit boards. In *Proc. Int. Conf. Comput.-Aided Des.* (pp. 713–720).

[40] Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2013c). Post-route refinement for high-frequency pcbs considering meander segment alleviation. In *ACM Great Lakes Symposium on VLSI* (pp. 323–324).

[41] Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2015a). ILP-based alleviation of dense meander segments with prioritized shifting and progressive fixing in pcb routing. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34(6), 1000–1013.

[42] Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2015b). Reliability-aware synthesis for flow-based microfluidic biochips by dynamic-device mapping. In *Proc. Design Autom. Conf.* (pp. 141:1–141:6).

[43] Tseng, T.-M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2015c). Storage and caching: Synthesis of flow-based microfluidic biochips. *IEEE Des. Test. Comput.*, 32(6), 69–75.

[44] Tseng, T.-M., Li, B., Li, M., Ho, T.-Y., & Schlichtmann, U. (2016a). Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 35(12), 1981–1994.

[45] Tseng, T.-M., Li, B., Yeh, C.-F., Jhan, H.-C., Tsai, Z.-M., Lin, M. P.-H., & Schlichtmann, U. (2016b). Novel CMOS RFIC layout generation with concurrent device placement and fixed-length microstrip routing. In *Proc. Design Autom. Conf.* (pp. 101:1–101:6).

[46] Tseng, T.-M., Li, B., Yeh, C.-F., Jhan, H.-C., Tsai, Z.-M., Lin, M. P.-H., & Schlichtmann, U. (accepted). An efficient two-phase ilp-based algorithm for precise cmos rfic layout generation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*

[47] Tseng, T.-M., Li, M., Li, B., Ho, T.-Y., & Schlichtmann, U. (2016c). Columba: Co-layout synthesis for continuous-flow microfluidic biochips. In *Proc. Design Autom. Conf.* (pp. 147:1–147:6).

[48] Unger, M. A., Chou, H.-P., Thorsen, T., Scherer, A., & Quake, S. R. (2000). Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(5463), 113–116.

[49] Wang, J., Fan, H. C., Behr, B., & Quake, S. R. (2012). Genome-wide single-cell analysis of recombination activity and *de novo* mutation rates in human sperm. *Cell*, 150(2), 402–412.

[50] White, A. K., VanInsberghe, M., Petriv, O. I., Hamidi, M., Sikorski, D., Marra, M. A., Piret, J., Aparicio, S., & Hansen, C. L. (2011). High-throughput microfluidic single-cell RT-qPCR. *Proc. Natl. Acad. Sci.*, 108(34), 13999–14004.

[51] Wu, A. R., Hiatt, J. B., Lu, R., Attema, J. L., Lobo, N. A., Weissman, I. L., Clarke, M. F., & Quake, S. R. (2009). Automated microfluidic chromatin immunoprecipitation from 2,000 cells. *Lab on a Chip*, 9, 1365–1370.

[52] Wu, A. R., Kawahara, T. L., Rapicavoli, N. A., van Riggelen, J., Shroff, E. H., Xu, L., Felsher, D. W., Chang, H. Y., & Quake, S. R. (2012). High throughput automated chromatin immunoprecipitation as a platform for drug screening and antibody validation. *Lab on a Chip*, 12, 2190–2198.

[53] Yao, H., Ho, T.-Y., & Cai, Y. (2015a). Pacor: Practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips. In *Proc. Design Autom. Conf.* (pp. 142:1–142:6).

[54] Yao, H., Wang, Q., Ru, Y., Ho, T.-Y., & Cai, Y. (2015b). Integrated flow-control co-design methodology for flow-based microfluidic biochips. *IEEE Des. Test. Comput.*, 32(6), 60–68.

[55] Zhong, J. F., Chen, Y., Marcus, J. S., Scherer, A., Quake, S. R., Taylor, C. R., & Weiner, L. P. (2008). A microfluidic processor for gene expression profiling of single human embryonic stem cells. *Lab on a Chip*, 8(1), 68–74.