Project Report On

Design Automation for Sample Preparation using Programmable Microfluidic Devices

Submitted in requirement for the course

B.Tech. Project (CSN-400B)

of Bachelor of Technology in Computer Science and Engineering

Submitted By

Sandeep Pal	**Gautam Choudhary**	**Siraz Shaikh**
15114063	15114027	15114065
sanpal1997@gmail.com	gc.iitr@gmail.com	sirazsk.97@gmail.com

Under the supervision of

Dr. Sudip Roy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

ROORKEE- 247667 (INDIA)

April, 2019

Candidate's Declaration

We declare that our work presented in this report with title **"Design Automation for Sample Preparation using Programmable Microfluidic Devices"** towards the fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** submitted in **Department of Computer Science And Engineering, Indian Institute of Technology, Roorkee** is an authentic record of our own work carried out during period from **August 2018 to April 2019** under the supervision of **Prof. Sudip Roy**, Assistant Professor, Dept. Of CSE, IIT Roorkee.
The content of this report, has not been submitted by us for the award for any other degree of this or any other institute.

DATE: ..... SIGNED:

PLACE: ..... GAUTAM CHOUDHARY
(15114027)

DATE: ..... SIGNED:

PLACE: ..... SANDEEP PAL
(15114063)

DATE: ..... SIGNED:

PLACE: ..... SIRAZ SHAIKH
(15114065)

Certificate

This is to certify that the statement made by the candidate is correct to the best of my knowledge and belief.

DATE: .....

SIGNED: .....

PROF. SUDIP ROY
(ASSISTANT PROFESSOR)
DEPT. OF CSE, IIT ROORKEE

Acknowledgement

First and foremost, we would like to express our sincere gratitude towards our guide **Dr. Sudip Roy**, Assistant Professor, Department of Computer Science and Engineering, IIT Roorkee for his ideal guidance throughout the entire period. We want to thank him for the insightful discussions and constructive criticisms which certainly enhanced our knowledge as well as improved our skills. His constant encouragement, support and motivation were key to overcome all the difficult and struggling phases.

We would also like thank **Department of Computer Science and Engineering, IIT Roorkee** for providing computing resources of **CoDA Lab**, and other resources for the project work.

We also extend our gratitude to **Debraj Kundu**, for keeping us motivated and providing us with valuable insights through various interesting discussions.

We humbly extend our sincere thanks to all concerned persons who co-operated with us in this regard.

Abstract

During last three decades various kinds of microfluidic laboratory-on-chips (LoCs) have been experimentally demonstrated. Compared to such conventional application-specific LoCs, which has less number of reconfigurable on-chip components (modules), the new technology known as programmable microfluidic devices (PMD) can be used to develop general-purpose LoCs with full reconfigurability. As per the working principle of a PMD-based LoC, fluids can easily be loaded into and washed from a cell with the help of fluid flows from inlet to outlet of the microfluidic chip, whereas cell-to-cell transportation of discrete fluid segment(s) is not precisely possible. On a PMD-based LoC, the simplest mixing module to realize is four-way mixer consisting of a 2×2 array of cells working as a ring-like mixer having four valves. For any bioassay implementation on an LoC, sample preparation is an important step, which needs to be automated for higher accuracy compared to manual intervention. In this paper, we propose a design automation strategy to achieve the transport-free module binding of a mixing tree (a 4-ary tree representing the sequence of mix-split steps) on a PMD-based LoC for sample preparation using four-way mixers. Moreover, we provide a heuristic to modify the mixing tree to reduce the time of completion for sample preparation. Simulation results confirm that the proposed heuristic along with the module binding method outperforms the module binding for the state-of-the-art mixing trees. There is a significant reduction in bioassay completion time (by a factor of 2) in many cases while decreasing the actuations of highly actuated valves. On an average, for ratio sum of 256, the bioassay completion time, area and valve used reduces by 21%, 23% and 28%, respectively, with only 25% increase in actuation count on-average.

Contents

List of Figures

List of Tables

List of Abbreviations

CMF	Continuous Microfluidic Device
DAG	Directed Acyclic Graph
DMF	Digital Microfluidic Device
DoF	Degree of Freedom
FPVA	Fully Programmable Valve Array
GUI	Graphical User Interface
HDA	Heuristic Distribution Algorithm
ILP	Integer Linear Programming
LoC	Lab On a Chip
MEDA	Micro Electrode Dot Array
NTM	No Transport Mixing
PMD	Programmable Microfluidic Device

List of Symbols

Symbol	Description
$\mathbb{A}$	Actuation Count
$\mathbb{C}$	Cell Count
$\mathbb{T}$	Bio assay completion time
$\mathbb{V}$	Valve Count
$\mathcal{T}$	Mixing Tree
N	Set of Nodes
E	Set of Edges
$\mathcal{G}$	Grid Graph
$\mathcal{A}$	Assignment Set
$\mathcal{L}$	Ratio Sum
M_i	Mix Node in G
r_i	Reagent Node in G
$C_{x,y}$	Cell situated at (x,y)
$M_{x,y}$	Mixer Module situated at (x,y)
L_C	Set of Load cells
M_C	Set of Mix cells
W_C	Set of Wash cells

Chapter 1

Introduction

1.1 BioChips

1.1.1 What is a BioChip?

Microfluidic biochips are technology for manipulating nano to femto-litre volume of biochemical fluids on a chip area of few centimeter squares. Biochips allow laboratory procedures(bioprotocols) to be applied on samples and reagents with minimal quantities as compared to traditional procedures, hence the name Lab-on-a-Chip. Biochips ensure less likelihood of error due to minimal human intervention. Some of the laboratory procedures have been successfully demonstrated on microfluidic biochips by different research groups. [3]

1.1.2 Types of BioChips

Based on technological advances done so far, biochips can be classified in two classes, continuous-flow based microfluidic (CMF) biochips and digital microfluidic (DMF) biochips, refer Figure 1.1,1.2.
Digital microfluidic (DMF) chips exploit the electrowetting-on-dielectric (EWOD) to perform droplet dispense, transport, mixing and splitting on a two-dimensional array of electrodes.
Whereas, a continuous-flow based microfluidic (CMF) biochip contains microchannels, microvalves and micropumps for manipulation of different biochemical fluids on it in order to implement any desired bioprotocol. Among all currently available technolog-

ical advances, DMF and CMF have become very popular due to the fact that they are suitable for design automation techniques.

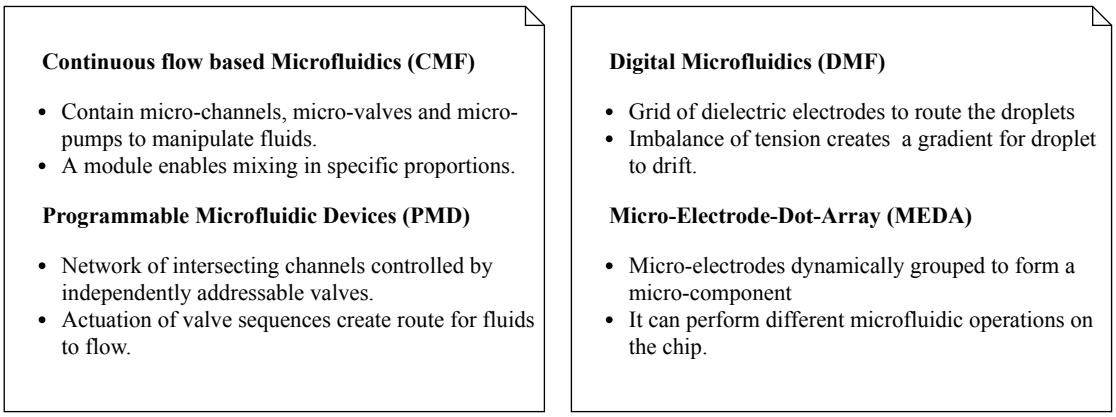
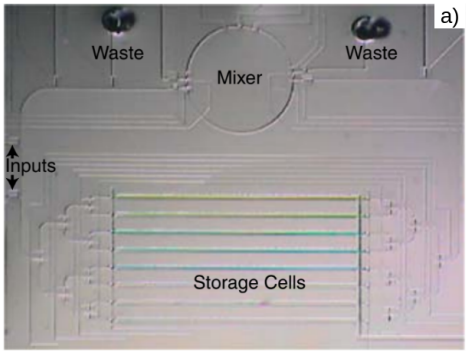


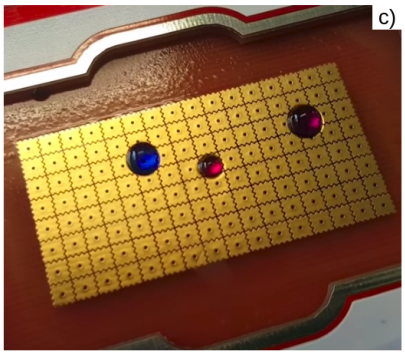
Figure 1.1: Two broad classes of biochips on basis of fluid driving force

Digital microfluidic biochips enable movement of fluid through a gradient in potential energy of cells. The droplet on the surface of the electrode shifts toward adjacent electrode when this electrode is actuated.

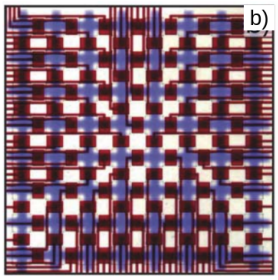
Continuous Flow based biochips enable movement of fluid through a pressure difference among chambers. As compared to digital microfluidics where droplet based operations are performed, in continuous microfluidics, a continuous flow of fluid is used. A buffer solution is used to push the fluid across chambers.



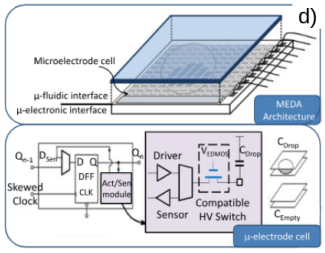
Source : William Thies et al, Nat Comput 2008



Source : OpenDrop Digital Microfluidics, (www.gaudi.ch/OpenDrop)



Source : Luis M. Fidalgo et al, Lab on a Chip, 2011



Source : Zipeng Li et al, IEEE TBCAS, 2017

Figure 1.2: Structures of different biochips : a) Continuous Microfluidic biochip, b) Prorgammable Microfluidic Device, c) Digital Microfluidic biochip and d)Micro-Electrode-Dot-Array biochip

1.1.3 Applications of BioChips

The increasing demand of safety-critical health-care applications makes people think about the need for automation of the expensive biochemical laboratory protocols in small-sized, hand-held and low-cost devices.

In contrast to macroscopic bio-systems, microfluidic biochips provide precise fluidic operations, consume less reactant fluids and accelerate the biochemical processes by parallelising certain steps of a bioprotocol. Thus the primary motivation for the design and synthesis of various microfluidc biochips is to integrate complex biochemical laboratory tasks in an efficient manner [4, 5]. To date several customized microfluidic biochips have been designed for automated drug discovery [6], prenatal testing [7], HIV and syphilis testing [8], DNA analysis [9, 10], cancer and stem cell research [11], environmental monitoring and biological weapons detection.

1.1.4 Problems Identified in BioChips

Biochips, along with its above mentioned advantages, also have some design and automation issues associated. Starting from the fabrication to on-chip synthesis, the following problems have been identified that are common to all kinds of biochips:

1. **Fabrication**: The fabrication in flow based devices has evolved from mLSI (microfluidic Large Scale Integration) to mVLSI (microfluidic VeriLarge Scale Integration) [12] where in the density of control valves has increased by a factor of 100. In the digital microfluidics, the technology has advanced from DMF (Digital Microfluidic Devices) to MEDA (Micro Electrode Dot Array) [13, 14] .

2. **High Level Synthesis (Scheduling)**: Scheduling determines when a micro operation of the bioassay starts and end execution on the microfluidic chip, while sticking to the precedence constraints (from the Directed Acyclic Graph representing the bioassay) and resource constraints on the microfluidic device. It is an NP-complete problem and is not possible to guarantee legal schedules in all instances [15].

3. **Architectural Level Synthesis (Placement, Routing)**:

   (a) *Placement*: This determines the precise locations on the microfluidic chip where the micro operations of bioassay are to be performed. The placer must ensure appropriate spacing between operations to prevent cross-contamination. This is also an NP-complete problem [16, 17].

   (b) *Routing*: This determines the fluid transportation paths to be routed from *(i)* input reservoirs on the microfluidic device to the placed modules *(ii)* from one placed module to another and *(iii)* from a placed module to an output or waste reservoir. The router must ensure appropriate spacing between flows of other ongoing operations, with the objective of minimizing total transportation time [18].

4. **Pin Mapping**: The number of independent input pins used to control the electrodes in digital microfluidic "biochips" is an important cost-driver that needs to be minimized. Pin-constrained biochips reduce the wiring complexity, thereby lowering the cost [19].

5. **Security and Attacks**: Potential vulnerabilities in microfluidic biochip give rise to security concerns with serious consequences for laboratory analysis and healthcare. Attacks may maliciously alter the intermediate concentration of some samples leading to alteration of bioassay final results. Some piracy attacks related to Digital Microfluidic Devices are also prevelant [20].

1.2 Objectives

The bioprotocols are represented as DAG with nodes representing the operations and edges representing the sequence. In order to execute these bioprotocols on PMD based biochips, we need to determine schedule and placement of reagents. We propose a placement algorithm to complete a bioassay on PMD based chip. Along with the algorithm, we introduce a heuristic that improves upon the algorithm to minimize the time and space for completion of the bioassay.

1.3 Organization

Having introduced biochips, their different types and their applications in **Chapter 1**. **Chapter 2** discusses about the literature survey and prior works done in this field. **Chapter 3** contains the proposed methodology and overview of our approach. **Chapter 4** discusses about the details of proposed algorithm for module binding. **Chapter 5** discusses the proposed heuristic for accelerating our algorithm. **Chapter 6** demonstrates the results and analyses the performance of the proposed algorithm. **Chapter 7** illustrates the GUI tool built for simulating the biochip. **Chapter 8** lays out the concluding remarks and extensions to our work.

Chapter 2

Literature Survey

2.1 Background on Programmable Microfluidic Devices

2.1.1 PMD: A Survey

Programmable Microfluidic Device (PMD) or Fully-Programmable Valve Array (FPVA) is a continuous flow-based microfluidic system, which consists of a network of microchannels controlled by an array of independently addressable microvalves [21]. FPVA-based LoC provides a generic platform, where we can perform fluidic operations such as mixing, dilution, and storage in a re-configurable fashion to implement different biochemical assays and hence, it serves as a general purpose microfluidic biochip. Fidalgo *et al.* [21] demonstrated bioprotocols like surface immunoassays, cell culture and active mixing on the FPVA biochip.

In literature, recently several research articles have been reported on the design automation and testing techniques for this similar kind of microfluidic biochips, referred as PMD-based or FPVA-based LoCs. In [22], an ILP (integer linear programming) based technique is proposed for a valve-centered architecture to make a reliable microfluidic system. A fault-tolerant and efficient multi-channel control logic minimization method is presented in [23] for flow-based microfluidic biochip, which can be efficiently implemented for PMD as well. This algorithm empowers PMD with faster parallel on-chip processes. A congestion avoiding routing algorithm for PMD is proposed in [24]. In [25] a more practical constraint, a pump aware flow routing algorithm is presented for PMD. An ILP based testing method for PMD is proposed in [26]. In [27] a relatively

time efficient spanning-tree based fault testing for PMD chip is reported. For a collision free routing an exact approach is presented in [28] where a valve sequence is generated as an output. A new dilution algorithm and its mapping for PMD is reported in [29]. Sample preparation is a very common step in many biochemical operations and can be achieved via sequence of mixing steps.

2.1.2 PMD: Applications

The reconfigurable architecture of Programmable Microfluidic Devices (PMD) has various potential use cases as shown in [30] and listed below:

1. **Microfluidic Display**: The device can be used for automated node addressing and compound loading. Simple programs have been created to generate dye patterns resembling letters. For creating each pattern, the program determines whether the node needs to be "re-written" into a new state: red or white.

2. **Surface Immunoassays**: Immunoassays constitute a fundamental tool in biological research, and their miniaturization has received significant attention. In their first experiment, they have used sequential channel routing across the network to create a surface pattern of a fluorescently labeled anti-GST (A-GST) antibody. In their second experiment, they create pattern using two antibodies, the same fluorescently-labeled anti-GST and a non-labeled anti-GFP (A-GFP) antibody.

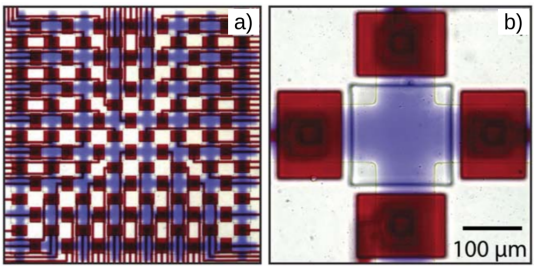
3. **Cell Culture**: Cell culture is another appealing application of microfluidics. The match between the dimensions of microfluidic channels and a wide variety of cell types combined with high-throughput and compatibility with light microscopy has made miniaturized cell culture a major focus in the field. Various specialized devices have been developed to culture bacteria, yeast and mammalian cells.

2.2 Basic Preliminaries of Programmable Microfluidic Devices

2.2.1 PMD: Architecture

PMD-based LoC is a two-layer microfluidic large-sacle integration (mLSI) based symmetric system. It consists of a network of intersecting flow-channels and control-

channels as shown in Figure 2.1. PMD-based LoCs are represented as a matrix/grid of cells of size $W \times H$, where W is the number of column and H is the number of rows in that matrix. Each of these cells are surrounded by four elastomeric valves each of which has a dedicated control address and can be programmed to generate a desired flow path. In a $W \times H$ PMD-based LoC the total number cells is $W * H$ and that of valves is $2W * H + W + H$.



Source : Luis M. Fidalgo et al, Lab on a Chip, 2011

Figure 2.1: A PMD device. a) Grid structure of PMD with fluid filled in red. b) Structure of a unit cell in grid surrounded by 4 valves.

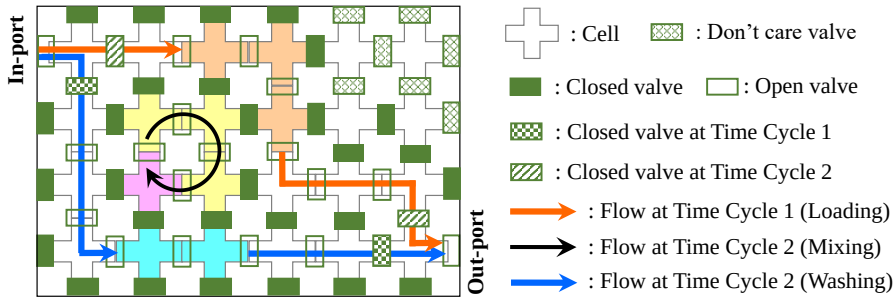


Figure 2.2: Schematic view of a PMD-based LoC and its working.

Loading, washing and mixing are the most atomic operations which are needed for the realization of any bioprotocols on a PMD-based LoC. As shown in Figure 2.2 at time cycle 1, both the loading and the mixing operations are preformed simultaneously and at time cycle 2, the washing operation is performed in the desired cells. Since any operation on a PMD-based LoC requires some specific actuation of valves the corresponding valve actuation states are shown in Figure 2.2. To load or wash a desired set

of cells with a particular fluid a flow-path is required from the in-port to the out-port which should pass through those desired cells such that it does not overlap with other concurrent flow-paths or obstacles. For mixing operation firstly the desired mixing fluids are loaded in a ring of cells then a set of consecutive valves of the corresponding cell ring are repeatedly actuated and released in a high frequency. This results in a peristaltic force and mix the fluids efficiently as in a rotary mixer [4].

2.2.2 PMD: Fluidic Transportation Constraint

As reported in literature, bubble formation and fluid segmentation in microchannels of a flow-based microlfuidic device is a well-known problem while transporting a reactant fluid from one part of the microchannel to another part with the help of another immiscible fluid (silicone oil) [31]. Same is the problem with PMD-based LoCs for transportation of a fluid from a set of source cells to some target cells as shown in Figure 2.3. In this work we consider a constraint which is not acknowledged before in the literature. To move a section of fluid reside inside some cells firstly a feasible flow path is determined and then using oil pressure from the in-port those cells are shifted to the target cells. As the flow channel is an orthogonal space there are several right-angle turns along the flow path. This results in a non-uniform oil pressure and as a consequence there may be a breakage in the initial fluid section. In Figure 2.3(a) two filled cells of the topmost row of the PMD chip is desired to shift in the the shaded cells of the bottom most row. The corresponding flow path is shown with a bold arrow. In Figure 2.3(b) the result after the oil pressure is shown where we can visualize the breaking of the moving fluid section. So here we discard the assumption of the feasible transport of a continuous fluid section.

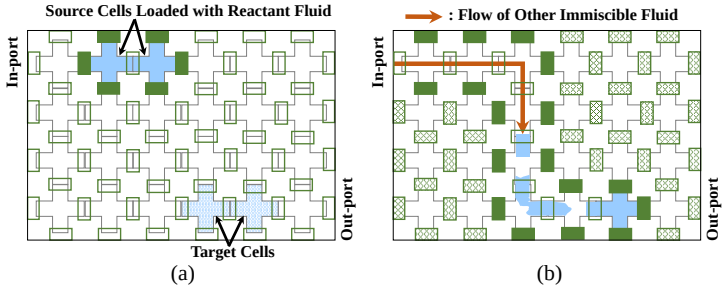


Figure 2.3: Fluid transport constraint for a PMD-based LoC. (a) Requirement of fluid transportation from source cells to target cells, and (b) fluid segmentation and bubble formation while transporting the fluid using another immiscible fluid.

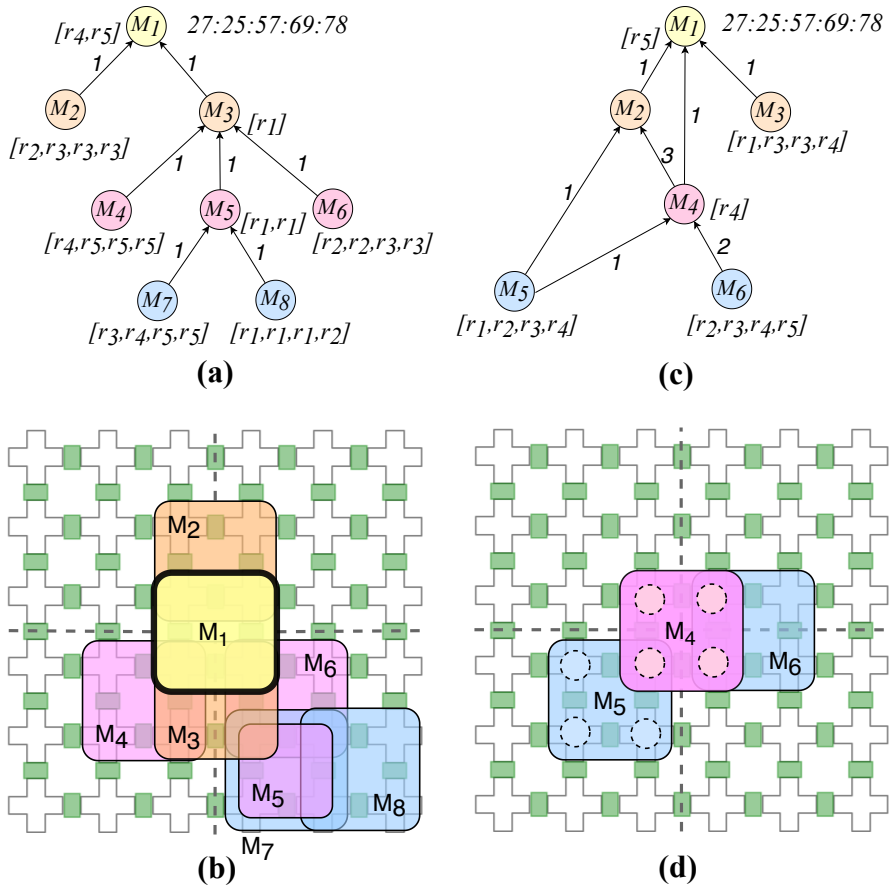


Figure 2.4: For target ratio $r_1 : r_2 : r_3 : r_4 : r_5 \equiv 27 : 25 : 57 : 69 : 78$, (a) mixing tree obtained by *genMixing* [1] and (b) its module binding using *NTM* in order $M_2, M_4, M_5 \rightarrow M_7 \rightarrow M_8 \rightarrow M_6 \rightarrow M_3 \rightarrow M_1$ with sequent washing, and (c) mixing graph obtained by *Storage-Aware FloSPA* [2] and (d) the problem raised during its module binding in order $M_5, M_6 \rightarrow M_4$.

2.2.3 Sample Preparation

Sample preparation is defined as the generation of a certain volume of mixed fluid which consists of particular ratio of k reagents (reactant fluids), $r_1, r_2, r_3, \cdots, r_k$. The automation process of such a desired ratio varies for different microfluidic technologies. In most of the cases the given ratio needs to be approximated into some other ratio set maintaining a error threshold termed as error tolerance ϵ. In general a sample preparation is formulated as:

Input: Target ratio $r_1 : r_2 : r_3 : \cdots : r_k \equiv a_1 : a_2 : a_3 : \cdots : a_k$, where $\sum_{i=1} a_i = \mathcal{L}$, error-tolerance ϵ.

Output: Mixing tree $\mathcal{T}(N, E)$, where N is the vertex set of the mixing tree. The non-leaf nodes represent different mixing operations whereas the leaf nodes represent the pure reagents. $\mathcal{T}(N, E)$ is basically a directed acyclic graph where the edges e ($e \in E$) defines the dependency among different mix operations. Many algorithms can be found in the literature to solve this problem [1, 32], the intermediate nodes of $\mathcal{T}$ represent mix-split or mix-separate operations and the leaf nodes act as pure reagents.

To realize a sample preparation on a microfluidic biochip, the task is to map the $\mathcal{T}$ into the biochip, i.e., mapping of intermediate nodes of a $\mathcal{T}$ with some on-chip microfluidic module (e.g., mixer). This problem is defined as application binding or module binding. The general convention is to schedule the mixing tree $\mathcal{T}$, then compute the feasible component binding and finally, the feasible placing of the modules. However, since we consider the fluidic transport constraint, we may get some unsolvable situations in the scheduling phase itself as shown in Figure 2.4 . So we simplified the problem and assumes: (a) only 2×2 mixer for performing any mixing operation, (b) left-factoring of the initial mixing tree Figure 4.4, (c) a postorder traversal of the left-factoring tree for the assignment problem. These assumptions reduce the corresponding scheduling and placement into deterministic problems.

Chapter 3

Proposed Methodology and Overview

The PMD cannot execute onchip movement of fluid, therefore the mix operations need to be performed without any movement of fluids. Thus, the fluids need to placed adjacently so that they are available for successive mix operations. Such a configuration may lead to deadlock and thus mix may not be possible. Therefore, we consider for our work only mixing trees with each node having incoming edge weight sum 4.
The mixing trees are placed on PMD using the algorithm we term as No-Transport Mixing(*NTM*) **Chapter 4**. The algorithm uses postorder traversal of tree to maintain chronological order of mix operations.
Later in **Chapter 5** we propose a heuristic for the modification of the initial mixing tree and then process the updated tree using NTM. This heuristic approach is coined as *Heuristic Distribution Algorithm (HDA)*. The heuristic is based on equipartition among different level of the tree. The pipeline of the whole process is summarised in Figure 3.1

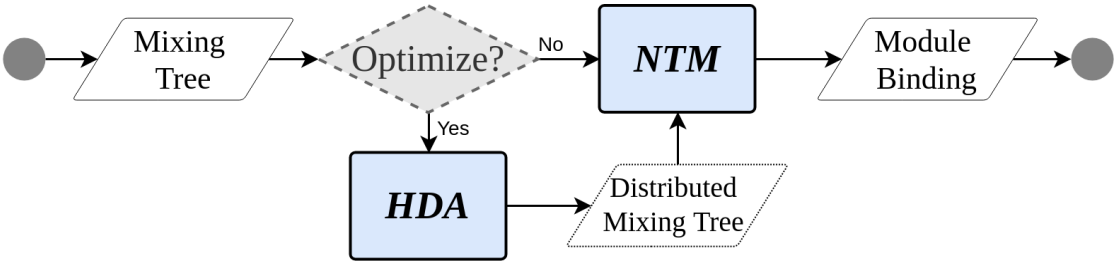


Figure 3.1: Proposed methodology for module binding in PMD-based LoC.

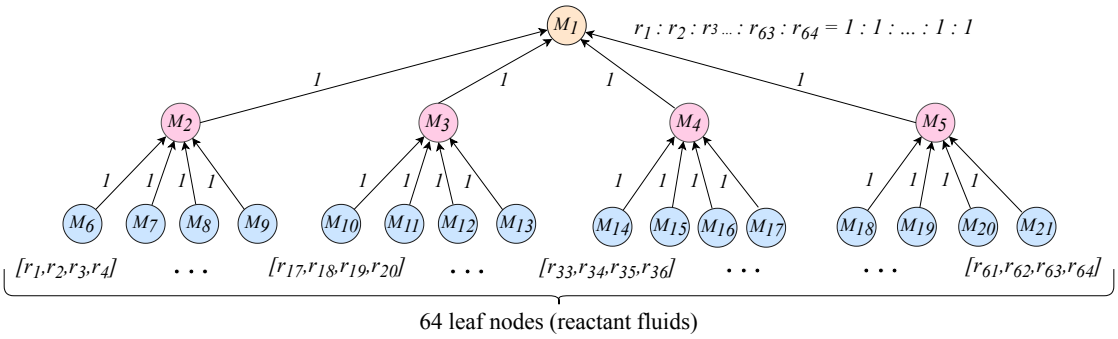
Chapter 4

Transport-Free Module Binding

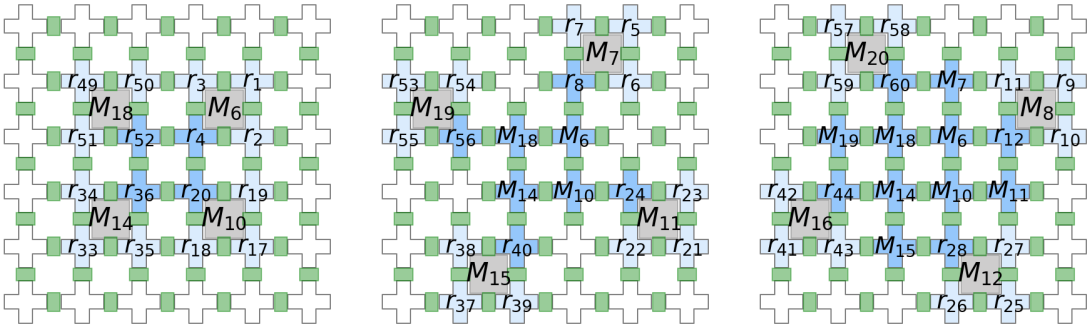
The transport constraint in PMD poses a big problem in mixture preparation. Here we propose a mixing algorithm with no cell-to-cell transportation of fluids namely *No Transportation Mixing* (*NTM*) algorithm.

4.1 Motivation

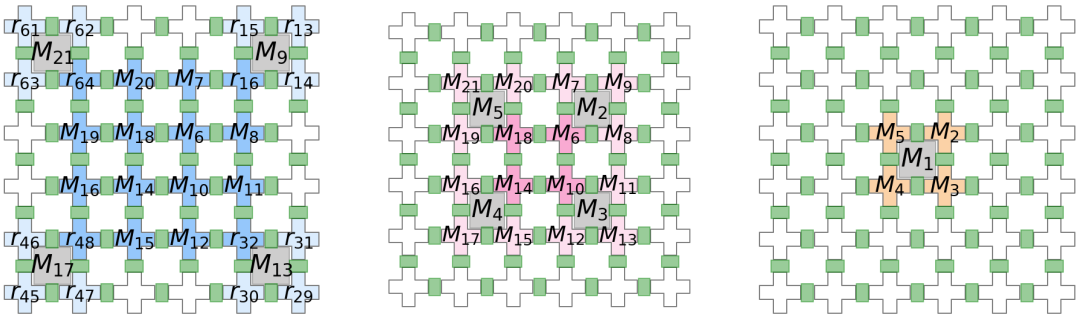
Sample preparation deal with a large set of mixing trees many of which have same structural skeleton. A node in a level of a tree is equivalent to mix of 4 same nodes in a level below. A pure reagent leaf node can be modeled as a mix node with that particular reagent in its lower level and also any reagent node can be permuted among themselves in a particular level [33]. So any mixing tree can be modelled as a complete tree. All the trees can thus be modelled as complete trees. Therefore, most general case of a tree is complete tree with all different reactants at the leaf nodes as shown in Figure 4.1. Any algorithm, able to bind a complete mixing tree with all different reactant fluids, will also be able to bind all the mixing trees structurally similar to it. The structurally similar trees, which is large in numbers can also be placed using same procedure and that too in less time and space. Thus, binding algorithm for the general case is the bottle-neck. We propose a solution for module binding as No Transport Mixing (*NTM*) Algorithm. *NTM* algorithm is developed in frame of, the most general case of complete mixing tree. *NTM* achieves the mix of complete tree with 64 different reactant fluids as shown in Figure 4.1. *NTM* always places the reactant fluids in a manner that they are always available for subsequent mixing operations.



(a)



(b) (c) (d)



(e) (f) (g)

Figure 4.1: (a) A mixing tree for target ratio $r_1 : r_2 : \ldots : r_{64} \equiv 1 : 1 : \ldots : 1$ obtained by *genMixing* [1] and module binding at time cycle (b) $t = 1$, (c) $t = 2$, (d) $t = 3$, (e) $t = 4$, (f) $t = 5$ and (g) $t = 6$.

4.2 Problem Formulation

The problem can be formulated as follows:
Input: *(i) Mixing tree* $\mathcal{T}(N : Nodes, E : Edges)$, where *indegree*$(n_i) = 4 \; \forall \; n_i \in N$ and *weight*$(e_i) = 1, 2$ or $3 \; \forall \; e_i \in E$. All internal nodes represent 'mix' nodes while all leaf nodes represent reagent fluids. *(ii) Grid graph* $\mathcal{G}(C : Cells)$, where the nodes $n_i \in N$ are arranged in a 2-D grid like structure with edges $e_i \in E$ connecting adjacent nodes.
Output: An assignment set $\mathcal{A} =$ *(TimeStamp, LoadCells, MixCells, WashCells)*. This set $\mathcal{A}$ is a binding sequence (mapping) of nodes of the tree $\mathcal{T}$ to the on-chip locations of the grid $\mathcal{G}$.
Objectives: To execute a mixing tree such that *(i)* chip area is **minimized** to ensure minimal space utilisation and *(ii)* mixes executing in parallel are **maximized** to ensure fast completion of bioassay.

4.3 Proposed Approach

4.3.1 Requirements

The proposed algorithm assumes the following requirements:

1. Inter/intra level sharing among mix nodes is not allowed.
2. Washing is done after every mix operation since the out degree of a mixing node can be $1, 2$ or 3.
3. The sum of incoming edge-weights of a mixing node should be 4.

4.3.2 Modelling

The following modelling is used for the proposed algorithm for module binding problem:

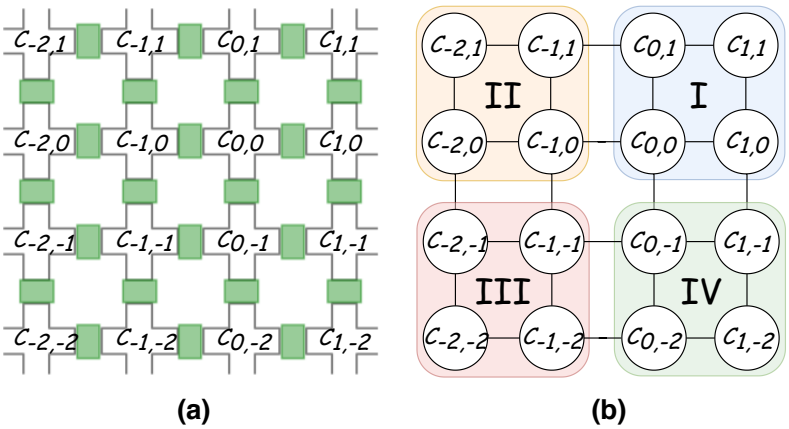


Figure 4.2: Illustration of proposed modelling for (a) PMD based LoC as a (b) 2-D grid graph.

1. **Mixing Tree** $\mathcal{T}(N, E)$: The sample preparation is an important part of many bioassays. This mixing could be modelled as a Tree where leaf nodes are reagents and internal nodes are intermediate mixture of those reagents or other mixtures.

2. **Grid Graph** $\mathcal{G}(\{C\})$: We model PMD based LoC as a $W \times H$ *grid* like 2-D structure as shown in Figure 4.2. The grid graph is composed of cells representing the usual points in 2-D plane. The chambers of PMD based LoC correspond to these cells in grid graph while the valves in PMD based LoC correspond to edges between adjacent nodes in grid graph which are not necessary for placement.

3. **Cells** $C_{x,y}$: A cell in grid graph is an ordered pair (x, y) in 2-D plane. Cell $C_{x,y}$ represents the location in grid graph where the final mixing is to be achieved.

4. **Module** $M_{x,y}$: A 2×2 mixer of PMD based LoC can be modelled as a *module*. It is composed of 4 cells $< C_{x,y}, C_{x-1,y}, C_{x-1,y-1}, C_{x,y-1} >$ representing the flow of mix. The ID of the mixer represents the top right corner of 2×2 module.

5. ***LOAD*(Module:** $M_{x,y}$**, ReagentNode:** R_i**)**: This operation of PMD, loads the reagent onto the module and returns the cell $C_{x,y}$ where the reagent R_i is loaded.

6. ***WASH*(Module:** $M_{x,y}$**, Volume:**V**)**: This operation of PMD, washes off V number of cells from the module and returns a set of washed cells $\{C_{x,y}\}$.

7. **Assignment Set** $\mathcal{A}$: This is a set of tuple (t : *TimeStamp*, L_C : *LoadCells*, M_C : *MixCells*, W_C : *WashCells*) representing the schedule and placement of mixing

tree as follows: At timestep 't', first load the cells with reagents as specified in L_C, then mix the cells specified in M_C and finally wash off W_C cells.

The mix corresponding to root node is performed in a 2×2 module having each cell in a different quadrant. We introduce the notion of parallelism as *degree of freedom*.

4.4 Degree of Freedom (DoF)

Degree of freedom for a mixing module is the number of its sub-mixes (or child mixes) that can be performed in parallel on the chip. Depending upon the sub-mixes of the root node, sub-mixes can be performed in parallel in each of the quadrant and thus has a degree of 4. Visualisation for DoF = 4, 2 and 1 is shown in Fig 4.3. In general, based on degree of freedom of the current mix node, the degree of freedom of the sub-mixes is determined.

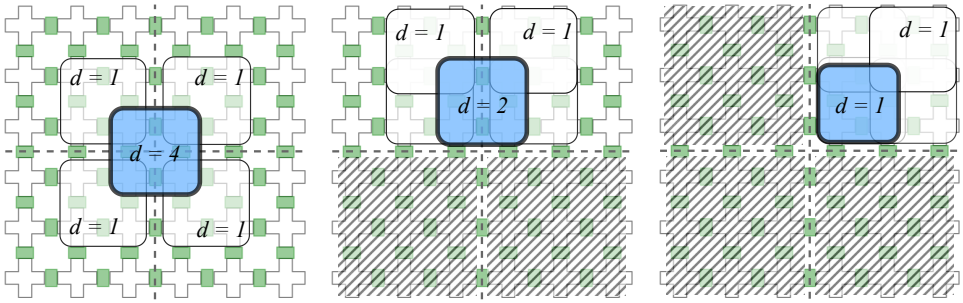


Figure 4.3: *Degree of Freedom* (DoF) in case of (a) $DoF = 4$, (b) $DoF = 2$ and (c) $DoF = 1$, where rounded squares represent parent mixer node (blue) and child mixer nodes (white) in a tree.

4.5 Left Factoring

There is a pre processing step involved before directly supplying input to our proposed algorithm. This is called *Left Factoring*. As the name suggests, this step transforms the tree in a format that all the child nodes are connected in decreasing order of their edge weigths with parent node. This is shown in Figure 4.4, where the mixing tree on left has node M_1 whose children are in order M_2, M_3. *Left Factoring* re-arranges the nodes

such that nodes with higher edge weights appear first in breadth first search traversal of tree. Hence, the left factored version (on the right of Figure 4.4) has M_3 appearing first and then M_2, and similarly M_5 appearing first and then M_4. This does not at all alter the mixing tree but only changes the order of children stored in the data structure. This method is realized using the *LeftFactoring* function in pseudocode.

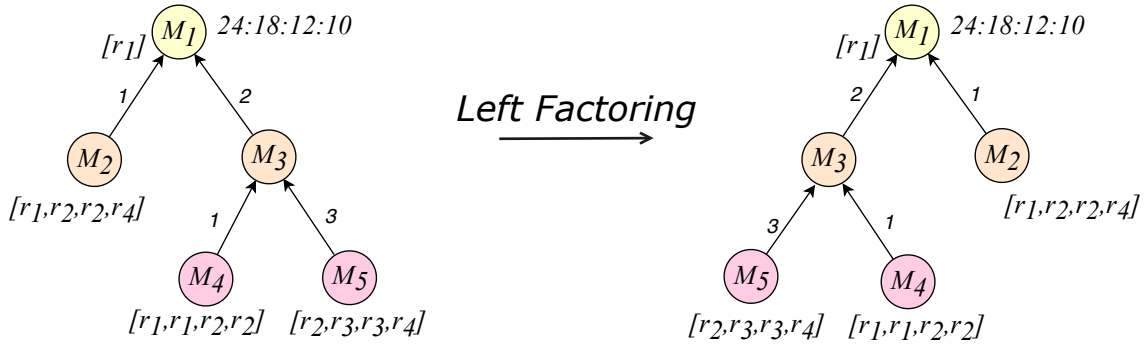


Figure 4.4: *Left Factoring* a mixing tree (left) shifts the higher edge weight carrying nodes to the left. The resultant is a left factored tree on right.

4.6 No Transport Mixing (*NTM*)

Starting from the root node, which is placed at the center of grid, i.e., origin, we assign its degree of freedom as 4. Based on number of mixer nodes in the root's children nodes, the degree of freedom is divided accordingly among the child-mixers. Now, the child-mixers are performed in parallel. The algorithm runs a post-order traversal of the sub-tree to ensure that a mix performs only when its sub-mixes have completed. Mix operation of current node is followed by a wash operation. The washing depends on how many fluid cells of that mix is used in its parent mix. For example, if the edge-weight of a mix is 1, then after mixing is done, 3 cells are washed off and only 1 is kept. This placement of mixing tree is executed in a recursive manner as written in **Algorithm 1**, named as *NTM*, until all the nodes are placed on the grid.

4.6.1 Example

Consider an example mixing tree $\mathcal{T}$ described in Figure 4.4 whose target ratio is $24 : 18 : 12 : 10$. The mixing tree is not necessary has to be a *genMixing* generated tree, it

can be any tree satisfying the constraints of input of our algorithm. Suppose we choose the coordinate axis for our PMD chip as described in Figure 4.2 and wish to place root node at $C_{0,0}$. The algorithm first computes the left factored tree $\mathcal{T}'$ and then placed the tree in the following manner:

1. Starting from the root node M_1 at first its DoF is distrubuted among its children M_3 and M_2. Since the DoF of M_1 is 4 so both M_3 and M_2 are assigned with equal DoF of 2 and the modules $M_{0,1}$ and $M_{0,-1}$ are assigned to M_3 and M_2 respectively.

2. M_2 has no children, so **Algorithm 2** directly loads the reagents in the module and performs the mixing operation and finally washes its 3 nodes since the out degree of M_2 is 1. This occurs at time stamp $t = 1$. Simultaneously M_3 places M_5 and the module $M_{0,1}$ is assigned to it as shown in Figure 4.5(a). The washed nodes are shown in light orange color.

3. After the completion of M_5, the cell $C_{-1,1}$ is washed off and the module $M_{-1,2}$ is assigned to M_4. Reagents required for M_4 are loaded and mixed and then 3 of its nodes are washed off, this happens at time $t = 2$ as shown in Figure 4.5(b).

4. At $t = 3$, reagents of M_3 are ready so the corresponding mixing, and then 2 node washing is done as shown in Figure 4.5(c) at $t = 3$.

5. Finally, the reagent r_1 is loaded and M_1 is now ready to execute. This is depicted in Figure 4.5(d).

This is how the placement algorithm works. There are cases to think of for example, how to assign Module to its children and deciding which mixes can happen in parallel and which require serial execution. The intrinsic details about this are skipped for simplicity purposes. A demonstration of the GUI based tool developed by us is available in our personal site, whose link is not included here to keep our identity secret at this initial submission stage, which follows a double-blind review process.

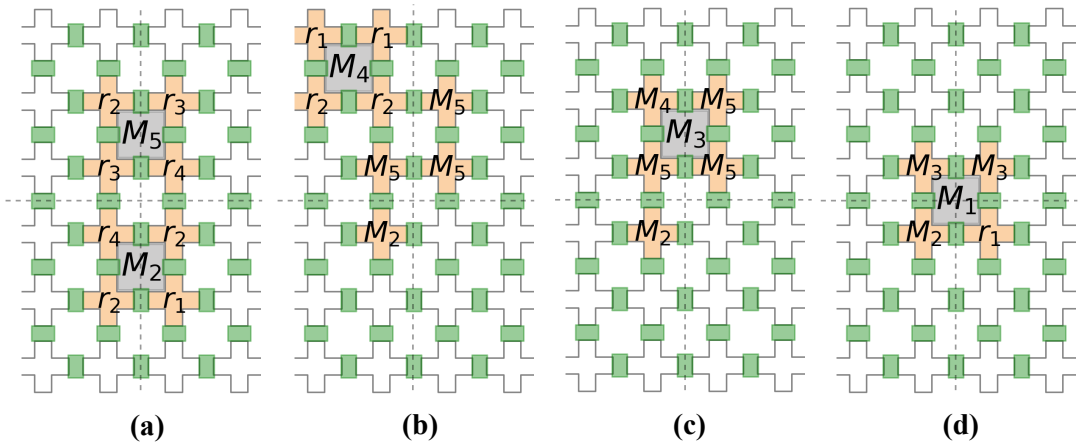


Figure 4.5: Module binding using *NTM* for left factored tree (as in Figure 4.4 at time cycle (a) $t = 1$, (b) $t = 2$, (c) $t = 3$ and (d) $t = 4$.

Algorithm 1: *placeNTM($\mathcal{T}$, $\mathcal{G}$, $C_{x,y}$)*

Input: Mixing Tree $\mathcal{T}(N, E)$, Grid Graph $\mathcal{G}\{C_{i,j}|i, j \in \mathbb{Z}^+\}$ and Cell $C_{x,y}$

Output: Assignment Set $\mathcal{A}$ of tuple (*TimeStamp* t, *LoadCells* L_C, *MixCells* M_C, *WashCells* W_C)

begin

/* Pre-processing the mixing tree $\mathcal{T}$ */

1 $\mathcal{T}' \leftarrow$ *LeftFactoring*($\mathcal{T}$);

/* Initialize Assignment Set */

2 $\mathcal{A} \leftarrow \phi$;

/* Place root node of $\mathcal{T}'$ at cell $C_{x,y}$ with $DoF = 4$ and starting from time step 1 */

3 **return** *placeHelper*($\mathcal{T}'$.*root*, $\mathcal{G}$.*module*($C_{x,y}$), 4, 1);

end

Algorithm 2: *placeHelper*($n, M_{x,y}, DoF, t$)

Input: Tree Node n, Mixer Module $M_{x,y}$, Degree of Freedom DoF, Time t
Output: Assignment Set $\mathcal{A}$ of tuple (*TimeStamp* t, *LoadCells* L_C, *MixCells* M_C, *WashCells* W_C)
begin
1 $\mathcal{A} \leftarrow \phi$; /* Initialize Assignment Set */
2 $L_C \leftarrow \phi$; $M_C \leftarrow \phi$; $W_C \leftarrow \phi$;
/* Recursively place every child of this mix node */
3 **for** $child_i$ *in* $n.children$ **do**
4 **if** $n.type ==$ *'MIX'* **then**
/* Place child mixers */
5 $M_{x_i,y_i} \subset \mathcal{G}$ s.t. $|M_{x_i,y_i} \cap M_{x,y}| \geq child_i$.outgoingEdgeWeight /* Corresponding DoF_i is computed for $child_i$ (refer sec. 4.4)*/
/* We keep a reference t_i to keep track of time taken by $child_i$ mix operation */
7 **if** $i == 0$ **then**
8 $t_i \leftarrow t$; // (pass-by-reference)
end
9 **else if** $child_i$ *mixes in parallel with* $child_{i-1}$ **then**
10 $t_i \leftarrow t_{i-1}$; // (pass-by-value)
end
else
11 $t_i \leftarrow t_{i-1}$; // (pass-by-reference)
end
13 $t_i \leftarrow max(t_i, t)$;
14 $\mathcal{A} \leftarrow \mathcal{A} \cup$ *placeHelper(*$child_i$, M_{x_i,y_i}, DoF_i, t_i*)*;
end
15 **else if** $n.type ==$ *'REAGENT'* **then**
/* Load child reagents in module $M_{x,y}$ */
16 $C_{x_i,y_i} \leftarrow$ *LOAD(*$M_{x,y}$, $child_i$*)*;
17 $L_C \leftarrow L_C \cup (child_i, C_{x_i,y_i})$;
end
end
/* Mix module $M_{x,y}$ */
18 $M_C \leftarrow (n, M_{x,y})$;
/* Wash specific volume of module $M_{x,y}$ */
19 $W_C \leftarrow$ *WASH(*M, $n.outgoingEdgeWeight$*)*;
/* Update Assignment Set $\mathcal{A}$ */
20 $\mathcal{A} \leftarrow \mathcal{A} \cup (t, L_C, M_C, W_C)$;
/* Increment the time counter */
21 $t \leftarrow t + 1$
22 **return** $\mathcal{A}$;
end

4.6.2 Complexity Analysis

To analyze the performance of our *NTM* algorithm, we present its time and space complexity in terms of number of nodes in mixing tree.

Time Complexity: Consider the worst case scenario of complete quaternary mixing tree of depth d, with depth of root node as $d = 0$. Then total time taken for the mixing can be found by the following relation

$$T = \sum_{i=0}^{d} \frac{4^d}{4}$$

This relation is derived from the basic principle that in 2×2 mixer, 4 mix operations can be performed in parallel. So, in each level, the total mix operations can be divided in groups of 4. Starting from lowest level of mixes in tree, where total mix nodes are 4^d (since it is at depth d), it takes $\frac{4^d}{4}$ mixing time steps (ignoring the time for loading/washing). Then, at depth $d-1$, it takes $\frac{4^{d-1}}{4}$ and similarly for upper levels. Summing up all the time steps in each level will yield the relation as shown above.

Space Complexity: On similar grounds, the worst case scenario of complete quaternary mixing tree of depth d, with depth of root node as $d = 0$. Then space consumed on grid for the mixing can be found by the following relation:

$$S = 2(d+1) \times 2(d+1) \quad \text{for } d = 1, 2, 3, ...$$

This relation can be easily derived looking at the placement of full quaternary mixing tree of depth d. Root node (depth, $d = 0$) is placed at center or origin i.e. its corner is at $(0, 0)$. At depth $d = 1$, the mix node placed farthest from origin is at $(1, 1)$. Similarly, the mix node at depth d is placed at (d, d). The total space consumed from 1st quadrant (starting from 0) is $(d+1) \times (d+1)$. Combining all 4 quadrants, the height H and width W of grid used will be $2(d+1) \times 2(d+1)$ as shown in Figure 4.6.

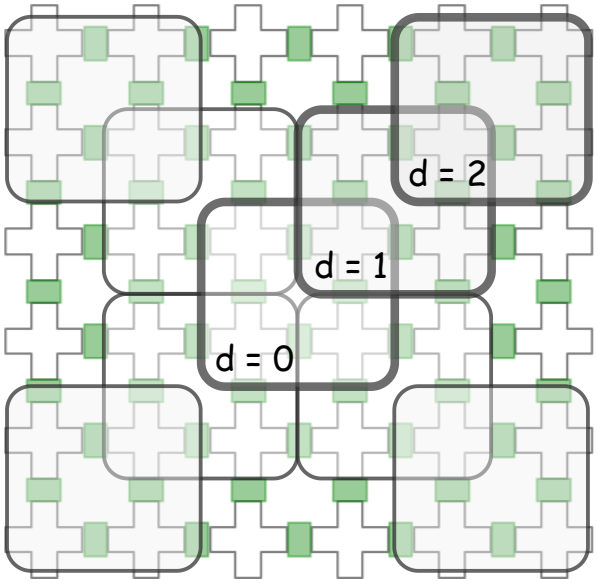


Figure 4.6: On chip placement, using *NTM*, of the farthest placed mixing nodes (rounded square boxes) present at varying depths $d = 0, 1, 2, ...$ of a general mixing tree.

Chapter 5

Mixing Tree Customization

The input mixing tree for *NTM* is a set of all the trees having, non-leaf nodes with incoming edge weight sum of 4. There may exist more than one tree having different structures but yielding same final-mix of the reagents. Such trees vary widely in their execution times according to *NTM*. Thus, it is to our advantage that we select the tree that possess minimum execution time and yields same final-mix.

5.1 Motivation

Even though *NTM* is able to bind mixing trees to PMD, it does not employ parallelism to fullest when there is possibility of achieving it through some other sequence of steps. Same mix ratio can be achieved through various mixing trees as shown in Figure 5.1(a), (b). The execution times of these mixing trees vary widely using *NTM*. The execution time for both the referred mixing trees are shown in Gantt charts in Figure 4.4(c), (d). Tree(a) completes its execution in 10 time cycles where as tree(b) completes in 4 time cycles. For tree(a), quad3 performs most of the mixes whereas they are evenly distributed for tree(b). The distribution of mixes can significantly improve the execution timing of *NTM*. *NTM* enables parallelism among mixers.These mixers need to perform in parallel in order to avoid overloading on some of them. Hence, it becomes a crucial problem to modify the given mixing-tree so that parallelism is maximized. We present a heuristic to improve the *NTM* algorithm in terms of time cycles required for completion of sample preparation.

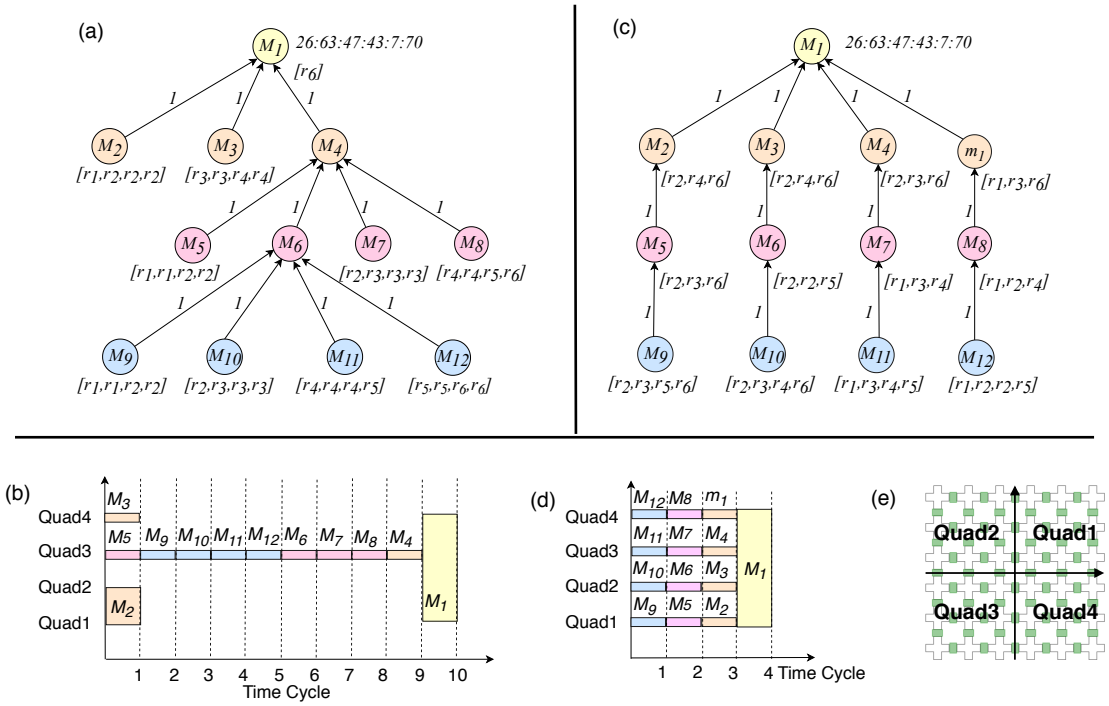


Figure 5.1: For a target ratio $26 : 63 : 47 : 43 : 7 : 70$ of six reactant fluids $r_1, r_2, r_3, r_4, r_5, r_6$, (a) a mixing tree obtained by *genMixing* [1] and (b) the Gantt chart showing its schedule, and (c) a modified mixing tree and (d) Gantt chart showing its schedule. (e) Four quadrants used for mixer module binding on an 8×8 PMD chip.

5.2 Problem Formulation

Input: A mixing tree $\mathcal{T}(N : Nodes, E : Edges)$, where *indegree*$(n_i) = 4\ \forall n_i \in N$ and *weight*$(e_i) = 1,\ \forall e_i \in E$. All internal nodes represent 'mix' nodes while all leaf nodes represent reagent fluids.

Output: A modified mixing tree $\mathcal{T}'(N' : Nodes, E' : Edges)$, where *indegree*$(n_i) = 4,\ \forall n_i \in N'$ and *weight*$(e_i) = 1\ \forall e_i \in E'$.

Objectives: To **minimize** the bioassay completion time without altering the final mix-ratio by modifying the tree. We try to equalize the size of each sub-tree.

5.3 Heuristic Distribution Algorithm (*HDA*)

We propose a heuristic to maximize parallelism among the mixers. The heuristic is for mixing-trees that have all the edge weights as 1 and indegree for all nodes as 4. But may also be extended for cases having edge weights not equal to 1, by replicating the sub-tree same number of times as its out-going edge weight. The heuristic is based on the principle, **the mixes in same level of a tree can be permuted [1]**. Based on the principle, we propose a method to permute mixes so that they are evenly distributed over the tree. The root of a mixing tree is placed at the center of the grid and hence, its child-mixes are performed in separate quadrants. Separate quadrants correspond to parallel mixing modules and we call them as **threads**. Therefore, we distribute mixes evenly among these threads.

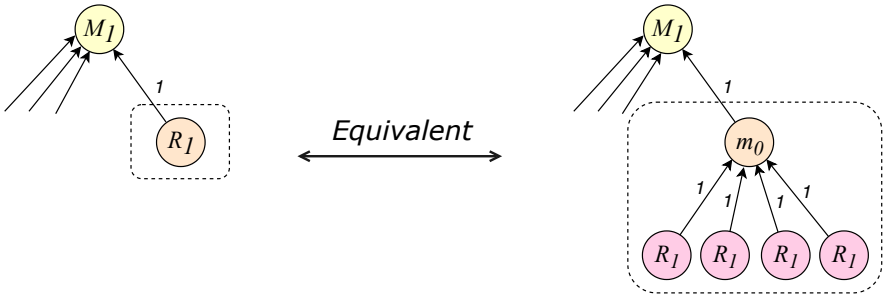


Figure 5.2: Substitution of a reagent R_1 with a dummy mix node m_0 and its 4 children R_1 results an equivalent of a mixing tree.

The pseudo-code of *HDA* written as **Algorithm 3** is explained below:

1. Mixers and reagents in i^{th} depth are represented by M_i and R_i, respectively. Maximum of M_i across all the depth is calculated.

2. *MAX_THREADS* is assigned the maximum of 4 or *maxMixers*. This represents the maximum number of *THREADS* that we need. Since, we cannot have more than 4 parallel mixers inside a 2×2 mixer, the *MAX_THREADS* cannot be more than 4.

3. We create as many number of *THREADS* as an array as *MAX_THREADS* for each level.

4. And join them into a bigger array Δ.

5. If for any depth of the tree, the number of mixes in the depth is less than threads, we add a dummy mix node at that depth corresponding to a reagent r.

6. Pop that reagent r.

7. And add 4 reagents of same type r in the reagent list R_{i+1} depth below it. *Basically, one unit of reagent in a depth is equivalent to 4 units of it in one depth below* as shown in Figure 5.2.

8. Then, the Mixes and Reagents in a depth are distributed evenly among all threads at same depth.

9. After this, the parent children connection reassignment step is done as shown in Figure 5.3. The mixes and reagents in corresponding threads at depth i are linked to corresponding threads at depth $i + 1$.

10. Each of thread corresponds to a tree and these threads are finally clubbed to create the new tree $\mathcal{T}'$.

Algorithm 3: *HDA($\mathcal{T}$)*

Input: Mixing tree $\mathcal{T}(N, E)$, where indegree($n_i \epsilon N$) = 4 and weight($e_i \epsilon E$) = 1
Output: Modified mixing tree $\mathcal{T}'(N', E')$, where indegree($n_i \epsilon N$) = 4 and weight($e_i \epsilon E$) = 1
begin
  /* M_i, R_i is list of mixers and reagents at depth *i* in tree $\mathcal{T}$ and *D* is depth of $\mathcal{T}$ where $D = 0, 1, 2, 3, \cdots$ */
1 $maxMixers \leftarrow max_{i=1}^{D}(M_i.size)$;
2 *MAX_THREADS* $\leftarrow min(4, maxMixers)$;
  /* Initialize thread set Δ where $\Delta_{i,j,k}$ represents k^{th} Mixer (m_k) of j^{th} thread at i^{th} Depth */
3 **for** *i:* $1 \rightarrow D$ **do**
    /* Create array of *MAX_THREADS* empty bins $B = [Bin_1, Bin_2, ...]$ */
4   Δ_i.*push(B)*;
  end
5 **for** *i:* $1 \rightarrow D$ **do**
    while M_i.*length* $<$ *MAX_THREADS* **do**
6      $M_i \leftarrow M_i \cup m_k$ /* Introduce new dummy mix nodes m_k; */
7      $r \leftarrow R_i$.*pop*();
8      $R_{i+1} \leftarrow R_{i+1} \cup \{r, r, r, r\}$;
    end
    /* Distribute all mixers $m_k \in M_i$ over all threads at i^{th} depth */
9   **for** *k:* $1 \rightarrow M_i$.*length* **do**
10      $x \leftarrow k \mod$ MAX_THREADS;
11      $\Delta_{i,x}$.*push*(m_k);
    end
    /* Distribute all reagents $r_k \in R_i$ over all threads at i^{th} depth */
    for *k:* $1 \rightarrow R_i$.*length* **do**
12      $x \leftarrow k \mod$ *MAX_THREADS*;
13      $\Delta_{i,x}$.*push*(r_k);
    end
  end
14 *Root* $\leftarrow \mathcal{T}$.*root*; /* Re-assign all node-children connection of given Tree $\mathcal{T}$ */
  /* Assign Root as parent node of $< \Delta_{1,1,1}, ..., \Delta_{i+1,MAX\_THREADS,1} >$ */
15 **for** *i:* $1 \rightarrow D - 1$ **do**
16   **for** *j:* $1 \rightarrow$ *MAX_THREADS* **do**
17      $L \leftarrow \Delta_{i+1,j}$.length;
18   **for** *k:* $1 \rightarrow \lfloor L/4 \rfloor$ **do**
          /* Assign $\Delta_{i,j,k}$ as parent node of $\Delta_{i+1,j,x}$ where $x = k, k+1, k+2, k+3$ */
        end
19      $k \leftarrow k + 1$;
        /* Assign $\Delta_{i,j,k}$ as parent node of $\Delta_{i+1,j,x}$ where $x = k, \cdots, L$ */
      end
    end
    return $\mathcal{T}$;
end

5.3.1 Example

Consider the mixing tree as shown in Figure 5.1(a) which is generated by *genMixing* [1]. To get equivalent tree according to *HDA*, for we proceed as follow :

1. First generate lists M_i and R_i for each depth.
2. Calculate *MAX_THREADS* which is 4 since none of the level has more than 4 mixing nodes. So, 4 threads of parallel mixers are created.
3. At first level since there are 3 mixers, but need to be 4. So create a dummy mixer m_1 which mixes 4 reagents of same type, here r_6. Then append M_1 with m_1 and also add 4 reagents of type r_6 to reagent list of the level below this depth.
4. Distribute all mixers and reagents at each depth in groups of 4 threads. Re-assign the child parent connection of nodes at depth i and $i+1$ as shown in Figure 5.3. The node at depth i is assigned parent of 4 consecutive nodes at depth $i+1$. Then the next node at depth i is assigned the next set of 4 nodes. We do this until all nodes of depth $i+1$ are assigned a parent in depth i.
5. Then move to next level and repeat the process until the last level where no assignment occurs.
6. Merge the 4 threads by assigning the top level node of each thread as child of the root node.

Figure 5.3: Parent children re-assignment step of depth i with depth $i+1$.

Chapter 6

Simulation Results

We consider a set of 50,000 or maximum possible (which ever is minimum) ratios for all ratio sum $\mathcal{L} = 64, 256, 1024$ and varying the number of reactant fluids $k = 3, 4, 5, \cdots, 12$. We perform two runs over the generated ratios to evaluate the parameter values. In the first run, we evaluate parameter values without *HDA* and in the second run we evaluate parameter values with *HDA*.

6.1 Environment Setup

Simulations are performed on a computer with a 3.70 GHz Intel Xeon processor and 16 GB memory running 64-bit Ubuntu 16.04 operating system. The code was written in Python 3.7 within the Anaconda Environment Packages. Jupyter-Notebooks were used to maintain literate programming style.

6.2 Results and Analysis

The performance is analyzed on 4 parameters: $\mathbb{T}$ (Time cycles for completion), $\mathbb{C}$ (Number of cells used on grid), $\mathbb{V}$ (Number of valves used on grid) and $\mathbb{A}$ (Total actuations of valves for mixing).

6.2.1 Performance Evaluation Based on $\mathcal{L}$

Table. 6.1 represents the average values of the parameters over all the ratios having $\mathcal{L} = 64, 256, 1024$. The average time cycles required for executing different test cases reduces with *HDA*. The average number of cells required and average valves involved are also less with *HDA*. On the other hand the average actuations of valves over the testcases increase with *HDA* because *HDA* introduces new mix nodes and the number of valve actuations increases with the number of mix nodes.

Table 6.1: Comparison between *NTM* and *HDA + NTM* on the basis of Avg. Completion Time $\mathbb{T}_{avg}$, Avg. #Cells $\mathbb{C}_{avg}$, Avg. #Valves $\mathbb{V}_{avg}$ and Avg. #Actuations $\mathbb{A}_{avg}$ for $\mathcal{L} = 64, 256, 1024$.

$\mathcal{L}$ ***(d)***	***NTM***				***HDA + NTM***			
	$\mathbb{T}_{avg}$	$\mathbb{C}_{avg}$	$\mathbb{V}_{avg}$	$\mathbb{A}_{avg}$	$\mathbb{T}_{avg}$	$\mathbb{C}_{avg}$	$\mathbb{V}_{avg}$	$\mathbb{A}_{avg}$
64 (3)	4.23	13.66	18.01	24.82	3.08	11.96	15.47	28.56
256 (4)	5.09	13.20	17.76	27.49	3.99	10.16	12.65	34.45
1024 (5)	5.95	12.68	16.99	30.64	4.99	9.55	11.86	42.16

6.2.2 Performance Evaluation Based on k

Figure 6.1 depicts the performance of both runs on the 4 parameters, with $\mathcal{L} = 64$ and ratios having k reagents. The difference in avg. completion time is more prominent with increasing number of reagents Figure 6.1(a). The avg. number of cells and valves used are less for *HDA + NTM*, but the difference diminishes with number of reagents. The avg. number of actuations increase for *HDA + NTM*, and the difference increases with k, which represents more number of mix nodes introduced in the mixing tree of *HDA*.

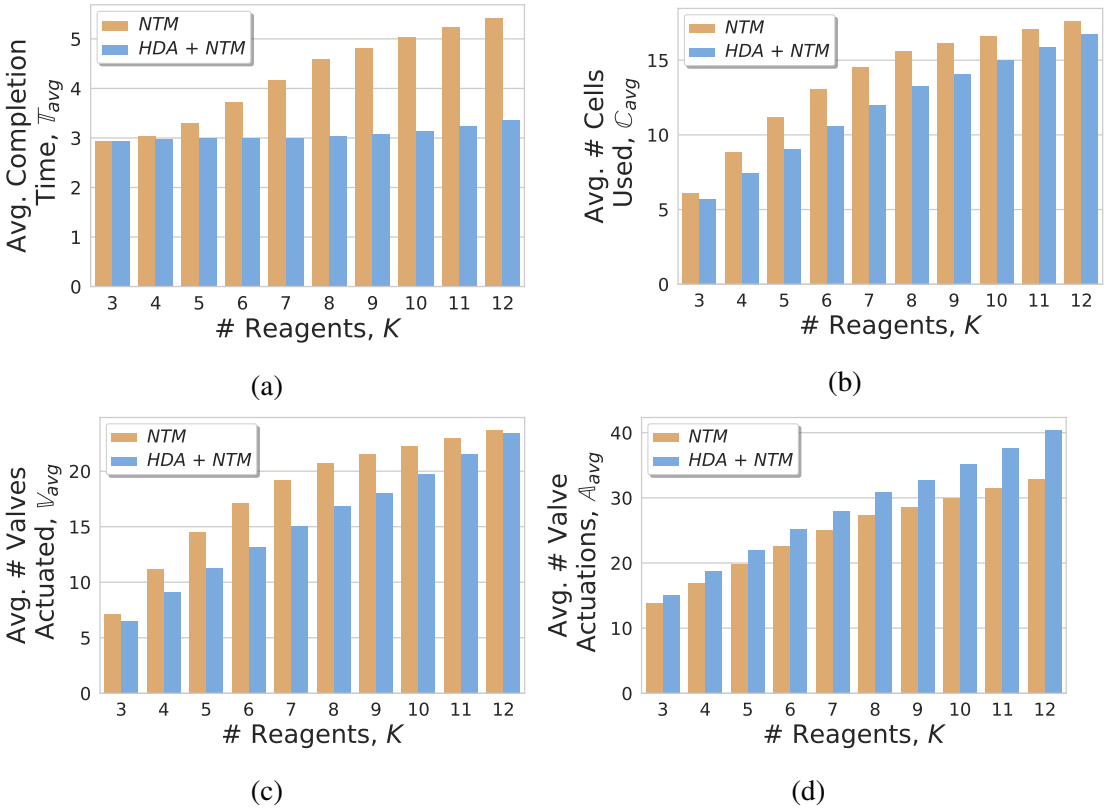


Figure 6.1: Distribution of (a) Avg. Completion Time $\mathbb{T}_{avg}$, (b) Avg. Cells $\mathbb{C}_{avg}$ used, (c) Avg. #Valves $\mathbb{V}_{avg}$ actuated and (d) Avg. #Valves $\mathbb{A}_{avg}$ actuations for ratio sum, $\mathcal{L} = 64$ $(d = 3)$ and over $k = 3, 4, 5, \cdots, 12$ for ***NTM*** and ***HDA + NTM***.

6.2.3 Performance Evaluation for Some Testcases

Table. 6.2 shows six random ratios from the testcases and their parameter values for both *NTM* and *HDA + NTM*. Time cycles and cells used for preparing the mix is always less with *HDA*. Total valves involved in the mix also decreases as the number of valves involved vary directly with number of cells involved. Total actuations of valves increase or remain same with *HDA*, it remains same when no new mix operations are introduced in *HDA* for all the test cases.

Table 6.2: Comparison between *NTM* and *HDA + NTM* on the basis of Completion Time $\mathbb{T}$, #Cells $\mathbb{C}$, #Valves $\mathbb{V}$ and #Actuations $\mathbb{A}$ for six different ratios.

#	Target Ratio	*NTM*				*HDA + NTM*			
		$\mathbb{T}$	$\mathbb{C}$	$\mathbb{V}$	$\mathbb{A}$	$\mathbb{T}$	$\mathbb{C}$	$\mathbb{V}$	$\mathbb{A}$
1	1:1:1:1:3:59:63:127	9	20	28	44	4	16	20	52
2	2:4:6:11:11:15:15	6	21	28	36	3	16	20	36
3	3:63:63:127	8	19	26	40	4	12	15	40
4	1:97:159:767	7	16	22	36	5	8	10	36
5	3:63:78:112	5	14	19	28	4	8	10	28
6	2:3:3:3:4:4:45	4	14	19	24	3	12	15	28

6.2.4 Comparative Heatmaps for Valve Actuations of a Testcase

The distribution of number of actuations on valves is also an important measure to analyze the algorithm's efficiency. The number of valves exceeding the threshold value are prone to breakage. Figure 6.2 depicts the heatmap for number of actuations of each valve. The heatmap is for the two mixing trees given in Figure 5.1(a) and Figure 5.1(c). The distribution of actuations on valves in case of *HDA + NTM* is evenly distributed, where as it varies largely for *NTM*.

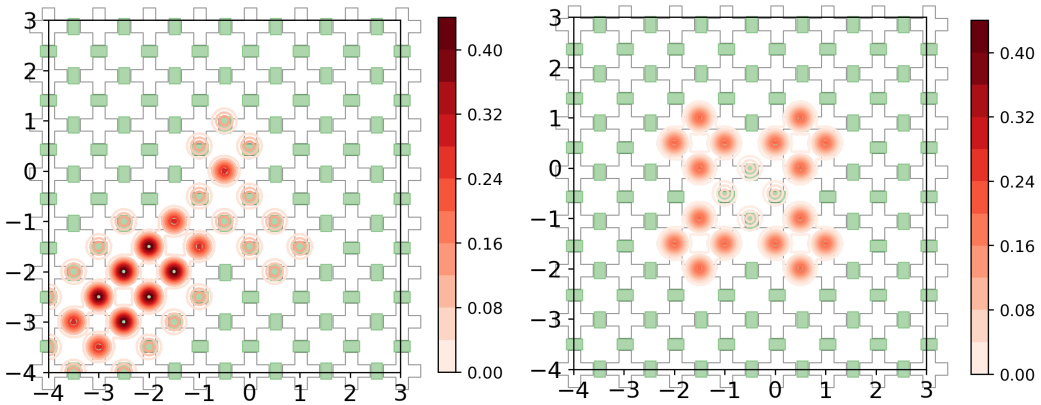


Figure 6.2: Heatmap for number of actuations of valves on the 8×8 PMD chip for (a) the mixing tree shown in Fig. 5.1(a) and (b) the mixing tree obtained by *HDA* shown in Fig. 5.1(c), after module binding performed by *NTM* algorithm.

Chapter 7

GUI Based Simulation Tool

We have developed a simulator to visualize the placements on the chip. The tool maps the coordinates assigned for placement on chip to the grid and shows the state of PMD chip after every time slice.

The PMD grid is visualized as shown in Figure 7.1. The fluid chambers are represented as white plus signs and the valves are represented as green bars. The left-bottom is inlet and top-right is outlet for the PMD. The valves are shown green, when in closed state and pale orange when in open state. The chamber represents the fluid it is filled with, here 'b' represents the buffer filled in all the chambers. The path from inlet to outlet for loading a fluid is represented by continuous grey cells.

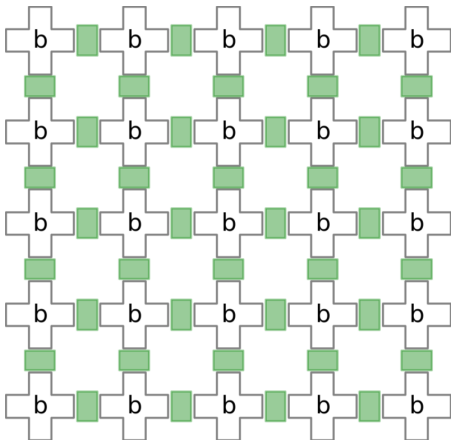


Figure 7.1: A snapshot of the tool at the beginning of execution

7.1 Implementation Details

The visualization tool has been developed using *matplotlib* library in Python. The module takes the placement information generated by NTM, and generates a series of snapshots of the state of the device using the flow shown in Figure 7.2.

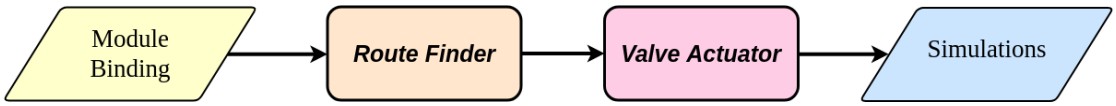


Figure 7.2: Flow for simulating the placement

The tool creates simulations from the placement information generated by *NTM*. At first the route from inlet to placed cells and further to outlet cell is determined. A continuous sequence of cells is determined along which the fluid needs to be flown in order to achieve the placed fluid at its designated position. The valves that need to be switched are determined from this path and the mix operations executing at the current time. The simulator marks the valves from the status determined in previous step. The state of valves and fluid is plotted using the simulator in every time slice and time slice corresponds to time interval for movement from one cell to any adjacent cell.

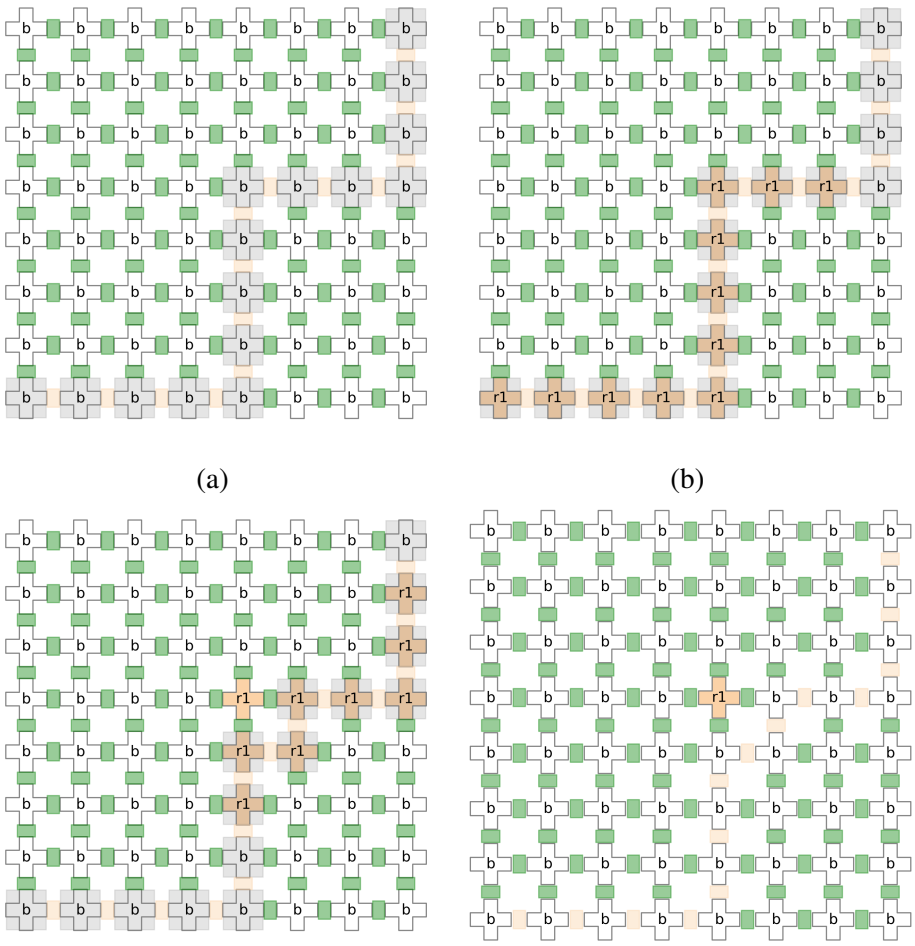
We have created a class of *Grid* with initalization code as described below. The grid is assumed to be in the first quadrant with origin at its bottom left corner. The mixer cycles denotes the number of cycles needed for mixing. Some helper functions like _plot_grid(), etc. are also used.

```
class Grid:
    def __init__(self, ID, inlet, outlet, height, width,
                 buffer='b', mixer_cycles=5):
        self.ID = ID
        self.INLET, self.OUTLET = inlet, outlet
        self.HEIGHT, self.WIDTH = height, width
        self.BUFFER = buffer
        self.MIXER_CYCLES = mixer_cycles

        self.G = self._init_graph()
        self.timecycle = 1
```

```
12
13          ### Take a Snapshot of initial PMD Grid ###
14          self._plot_grid(self.G)
15          return
16
```

7.2 Example

7.2.1 Load Operation



(a) (b)

(c) (d)

Figure 7.3: Load Operation in GUI tool

The Load operation as displayed via sequences of snapshots in Figure 7.3 has been coded as described below.

1. First, we obtain a route from inlet to outlet that passes through the cells which need to be loaded, here (4, 4). Here, in Figure 7.3(a), the cells highlighted in gray represent the path from inlet, which is at (0, 0), to outlet, which is at (7,7) through (4, 4).

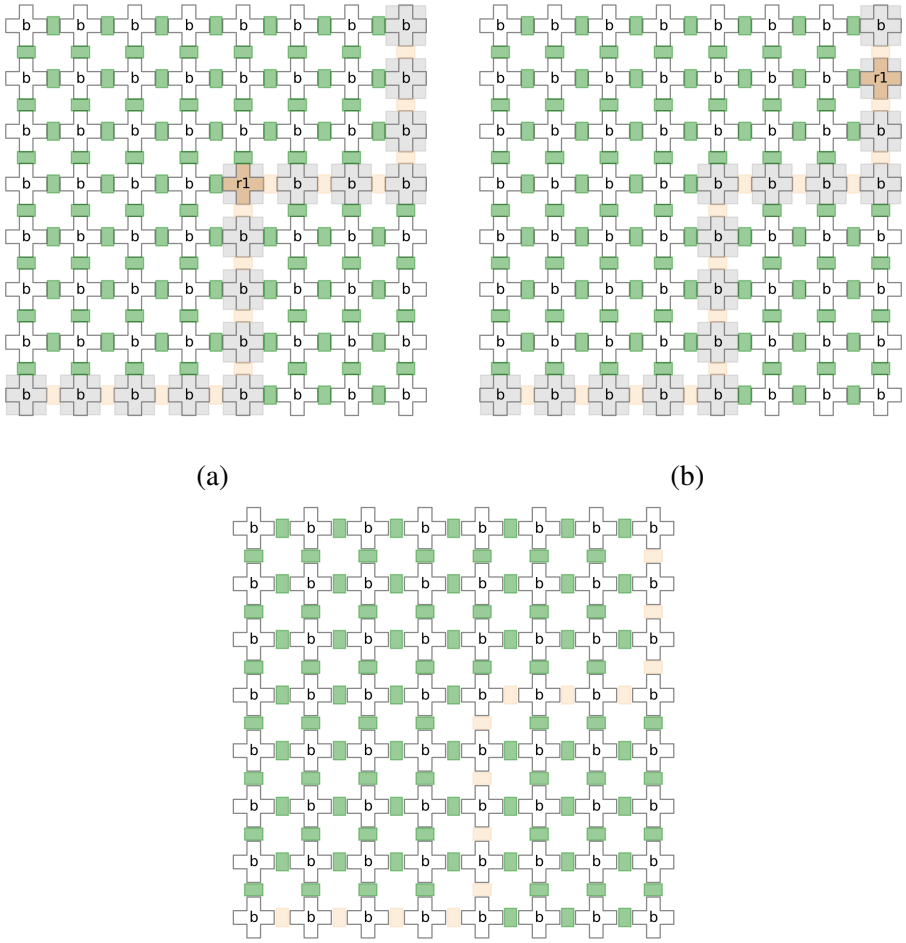
2. The required fluid, here $r1$, is flown through *load_route* as shown in Figure 7.3(b).

3. Another route, *wash_route*, is also obtained which includes all cells of *load_route* but excludes cells which need to be loaded, here (4, 4). The cells highlighted in gray in Figure 7.3(c) represent the path for *wash_route*. BUFFER SOLUTION is flown through *wash_route* to remove the un-necessary fluid.

4. Finally, the required cells, here (4, 4), are loaded with fluid $r1$ and the snapshot of the grid is saved to disk after the Load operation.

```
class Grid:
    ...
    def Load(self, fluid, node_list):
        # Find the route from inlet to outlet through node_list
        # Find another route same as prev. but excluding node_list
        load_route, wash_route = self._find_route(self.G, node_list)

        # Flow the fluid along load_route
        self._flow(self.G, fluid, load_route)

        # Mark node_list as 'used'
        self._mark_nodes(self.G, used=node_list, unused=[])

        # Flow the buffer along wash_route
        self._flow(self.G, self.BUFFER, wash_route)

        ### Take a Snapshot after Load operation ###
        self._plot_grid(self.G)
        return

```

7.2.2 Wash Operation



(a) (b)

(c)

Figure 7.4: Wash Operation in GUI tool

The Wash operation as displayed via sequences of snapshots in Figure 7.4 has been coded as described below.

1. First, we obtain a route, *wash_route*, from inlet to outlet that passes through the cells which need to be washed, here (4, 4). Here, in Figure 7.4(a), the cells highlighted in gray represent the path from inlet, which is at (0, 0), to outlet, which is at (7,7) through (4, 4).

2. BUFFER SOLUTION is flown through *wash_route* to wash of the required cells, here (4, 4) as shown in Figure 7.4(b).

3. Finally, the required cells, here (4, 4), are washed off and the snapshot of the grid is saved to disk after the Wash operation.

```
class Grid:
    ...
    def Wash(self, node_list):
        # Find the route from inlet to outlet through node_list
        wash_route, _ = self._find_route(self.G, node_list)

        # Flow the buffer along wash_route
        self._flow(self.G, self.BUFFER, wash_route)

        # Mark wash_route as 'un-used'
        self._mark_nodes(self.G, used=[], unused=node_list)

        ### Take a Snapshot after Wash operation ###
        self._plot_grid(self.G)
        return

```

7.2.3 Mix Operation

The Mix operation as displayed via sequences of snapshots in Figure 7.5 has been coded as described below.

1. For each mixer module in *mixer_list*, we actuate corresponding valves. For each time cycles, the valves inside each mixer module are actuated periodically.

2. Finally, the snapshot of the grid is saved to disk after the Mix operation.

```
class Grid:
    ...
    def mix(self, mixer_list):
        for i in range(self.MIXER_CYCLES):
            # Actuate required valves for each mixer at i^th time
    cycle
            ...
```

```
            self._actuate(valves_open_i, valves_close_i)

            # Take a Snapshot at i^th time cycle
            self._plot_grid(self.G)
        # Take a Snapshot after MIX operation
        self._plot_grid(self.G)
        return

```

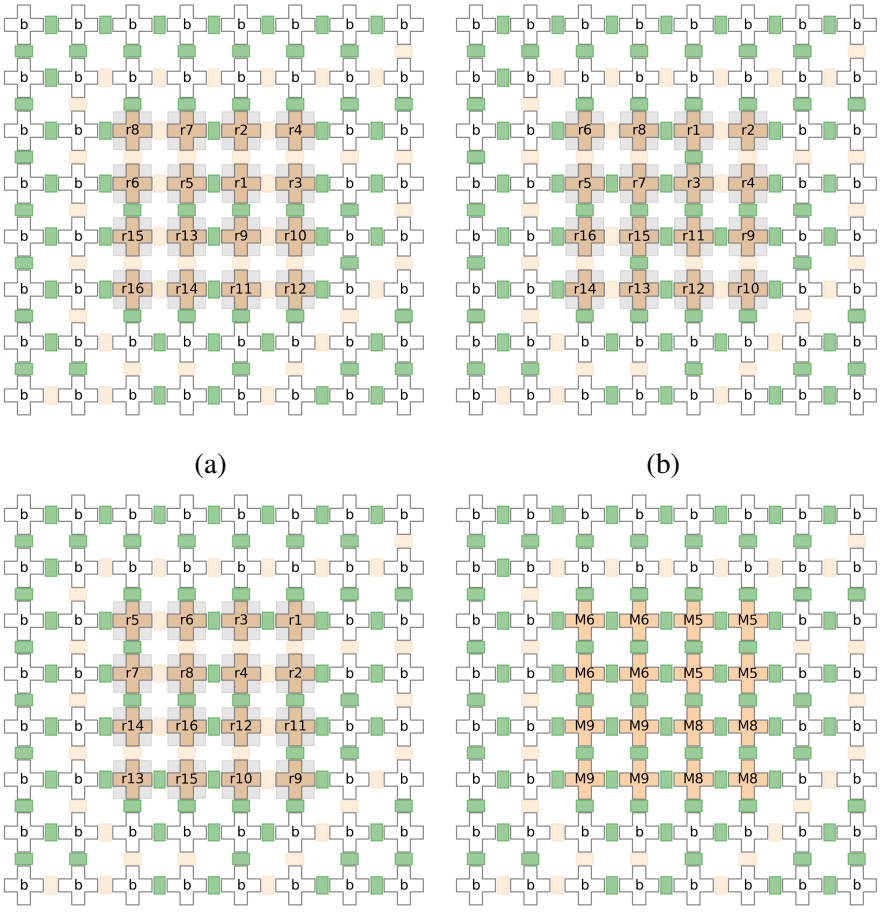


Figure 7.5: Mix Operation in GUI tool

Chapter 8

Conclusions and Future Work

A lot of work has been done on the automation of sample preparation for different microfluidic technologies.In this paper, we present an algorithm for binding of such application on a PMD chip. The fluid-section transport problem is defined and we model the proposed algorithm considering this technology specific constraint. The proposed binding method can realize any mixing tree using optimal number of PMD cells. A heuristic is also proposed to modify a mixing tree generated using *genMixing* for mixer size equals to four. The mixing operations of the modified mixing tree can be distributed in an efficient concurrent fashion and helps in reducing the overall operation time. The modified mixing tree also able to achieve a more reliable PMD chip by distributing the valve actions.

The future work may be based upon the extension of placement algorithm for a general N-way mixer. Another possibility is the extension of our heuristic algorithm (3) to work for edge weights > 1 in the mixing tree.

Dissemination from the Dissertation

Gautam Choudhary, Sandeep Pal, Debraj Kundu, Sukanta Bhattacharjee, Shigeru Yamashita and Sudip Roy, "Transport-Free Module Binding for Sample Preparation Using Programmable Microfluidic Devices", **submitted to** *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* to be held in Colorado, USA during November 04-07, 2019.

Bibliography

[1] S. Bhattacharjee, S. Poddar, S. Roy, J. D. Huang, and B. B. Bhattacharya, "Dilution and Mixing Algorithms for Flow-Based Microfluidic Biochips," *IEEE TCAD*, vol. 36, no. 4, pp. 614–627, 2017.

[2] S. Bhattacharjee, R. Wille, J. D. Huang, and B. B. Bhattacharya, "Storage-Aware Algorithms for Dilution and Mixture Preparation with Flow-Based Lab-on-Chip," *IEEE TCAD*, vol. (Early Access), pp. 1399–1404, 2019.

[3] F. Su and K. Chakrabarty, *Digital microfluidic biochips: synthesis, testing, and reconfiguration techniques*. CRC Press, 2006.

[4] W. Thies, J. P. Urbanski, T. Thorsen, and S. Amarasinghe, "Abstraction Layers for Scalable Microfluidic Biocomputing," *Natural Computing*, vol. 7, no. 2, pp. 255–275, 2008.

[5] Z. Li, K. Chakrabarty, T.-Y. Ho, and C.-Y. Lee, *Micro-Electrode-Dot-Array Digital Microfluidic Biochips: Design Automation, Optimization, and Test Techniques*. Springer, 2019.

[6] S. Einav, D. Gerber, P. D. Bryson, and *et. al.*, "Discovery of a Hepatitis C Target and its Pharmacological Inhibitors by Microfluidic Affinity Analysis," *Nature Biotechnology*, vol. 12, pp. 1019–1027, 2008.

[7] H. C. Fan, Y. J. Blumenfeld, U. Chitkara, L. Hudgins, and S. R. Quake, "Noninvasive Diagnosis of Fetal Aneuploidy by Shotgun Sequencing DNA from Maternal Blood," *Proceedings of the National Academy of Sciences*, vol. 105, no. 42, pp. 16 266–16 271, 2008.

[8] C. D. Chin, T. Laksanasopin, Y. K. Cheung, and *et. al*, "Microfluidics-Based Diagnostics of Infectious Diseases in the developing World," *Nature Medicine*, vol. 17, pp. 1015–1019, 2011.

[9] H. C. Fan, Y. J. Blumenfeld, U. Chitkara, L. Hudgins, and S. R. Quake, "Noninvasive diagnosis of fetal aneuploidy by shotgun sequencing dna from maternal blood," *Proceedings of the National Academy of Sciences*, vol. 105, no. 42, pp. 16 266–16 271, 2008. [Online]. Available: https://www.pnas.org/content/105/42/16266

[10] C. L. Hansen, M. O. A. Sommer, and S. R. Quake, "Systematic investigation of protein phase behavior with a microfluidic formulator," *Proceedings of the National Academy of Sciences*, vol. 101, no. 40, pp. 14 431–14 436, 2004. [Online]. Available: https://www.pnas.org/content/101/40/14431

[11] C. Fang, Y. Wang, N. T. Vu, W.-Y. Lin, Y.-T. Hsieh, L. Rubbi, M. E. Phelps, M. Müschen, Y.-M. Kim, A. F. Chatziioannou, H.-R. Tseng, and T. G. Graeber, "Integrated microfluidic and imaging platform for a kinase activity radioassay to analyze minute patient cancer samples," *Cancer Research*, vol. 70, no. 21, pp. 8299–8308, 2010. [Online]. Available: http://cancerres.aacrjournals.org/content/70/21/8299

[12] I. E. Araci and S. R. Quake, "Microfluidic Very Large Scale Integration (mVLSI) with Integrated Micromechanical Valves," *Lab Chip*, vol. 12, no. 16, pp. 2803–2806, 2012.

[13] G. Wang, D. Teng, and S.-K. Fan, "Digital Microfluidic Operations on Micro-Electrode Dot Array Architecture," *IET Nanobiotechnology*, vol. 5, no. 4, pp. 152–160, 2011.

[14] J. H. Noh, J. Noh, E. Kreit, J. Heikenfeld, and P. D. Rack, "Toward Active-matrix Lab-on-a-Chip: Programmable Electrofluidic Control Enabled by Arrayed Oxide Thin Film Transistors," *Lab Chip*, vol. 12, no. 2, pp. 353–360, 2012.

[15] K. Chakrabarty and F. Su, *Digital Microfluidic Biochips: Synthesis, Testing and Reconfiguration Techniques*. CRC Press, 2007.

[16] Z. Li, K. Y.-T. Lai, P.-H. Yu, K. Chakrabarty, T.-Y. Ho, and C.-Y. Lee, "Droplet Size-Aware High-Level Synthesis for Micro-Electrode-Dot-Array Digital Microfluidic Biochips," *IEEE TBCAS*, vol. 11, no. 3, pp. 612–626, 2017.

[17] Q. Wang, H. Zou, H. Yao, T. Ho, R. Wille, and Y. Cai, "Physical Co-Design of Flow and Control Layers for Flow-Based Microfluidic Biochips," *IEEE TCAD*, vol. 37, no. 6, pp. 1157–1170, 2018.

[18] O. Keszöcze, Z. Li, A. Grimmer, R. Wille, K. Chakrabarty, and R. Drechsler, "Exact Routing for Micro-Electrode-Dot-Array Digital Microfluidic Biochips," in *Proc. of the ASP-DAC*, 2017, pp. 708–713.

[19] Y. Zhao, T. Xu, and K. Chakrabarty, "Broadcast Electrode-Addressing and Scheduling Methods for Pin-Constrained Digital Microfluidic Biochips," *IEEE TCAD*, vol. 30, no. 7, pp. 986–999, 2011.

[20] M. Shayan, S. Bhattacharjee, Y. A. Song, K. Chakrabarty, and R. Karri, "Security Assessment of Microfluidic Immunoassays," in *Proc. of the COINS*, 2019, pp. 217–222.

[21] L. M. Fidalgo and S. J. Maerkl, "A Software-Programmable Microfluidic Device for Automated Biology," *Lab Chip*, vol. 11, pp. 1612–1619, 2011.

[22] T. Tseng, B. Li, M. Li, T. Ho, and U. Schlichtmann, "Reliability-Aware Synthesis With Dynamic Device Mapping and Fluid Routing for Flow-Based Microfluidic Biochips," *IEEE TCAD*, vol. 35, no. 12, pp. 1981–1994, 2016.

[23] Y. Zhu, B. Li, T.-Y. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann, "Multi-channel and Fault-tolerant Control Multiplexing for Flow-based Microfluidic Biochips," in *Proc. of the ICCAD*, 2018, pp. 123:1–123:8.

[24] Y. Su, T.-Y. Ho, and D. Lee, "A Routability-Driven Flow Routing Algorithm for Programmable Microfluidic Devices," in *Proc. of the ASP-DAC*, 2016, pp. 605–610.

[25] G. Lai, C. Lin, and T.-Y. Ho, "Pump-Aware Flow Routing Algorithm for Programmable Microfluidic Devices," in *Proc. of the DATE*, 2018, pp. 1405–1410.

[26] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T.-Y. Ho, and U. Schlichtmann, "Testing Microfluidic Fully Programmable Valve Arrays (FPVAs)," in *Proc. of the DATE*, 2017, pp. 91–96.

[27] A. Bernardini, C. Liu, B. Li, and U. Schlichtmann, "Efficient Spanning-tree-based Test Pattern Generation for Programmable Microfluidic Devices," *Microelectronics Journal*, vol. 79, pp. 38–45, 2018.

[28] A. Grimmer, B. Klepic, T.-Y. Ho, and R. Wille, "Sound Valve-Control for Programmable Microfluidic Devices," in *Proc. of the ASP-DAC*, 2018, pp. 40–45.

[29] A. Gupta, J. Huang, S. Yamashita, and S. Roy, "Design Automation for Dilution of a Fluid Using Programmable Microfluidic Device-Based Biochips," *ACM TODAES*, vol. 24, no. 2, pp. 21:1–21:24, 2019.

[30] L. M. Fidalgo and S. J. Maerkl, "A software-programmable microfluidic device for automated biology," *Lab on a Chip*, vol. 11, no. 9, pp. 1612–1619, 2011.

[31] T. Fu and Y. Ma, "Bubble formation and breakup dynamics in microfluidic devices: A review," *Chemical Engineering Science*, vol. 135, pp. 343–372, 2015.

[32] S. Saha, D. Kundu, S. Roy, S. Bhattacharjee, K. Chakrabarty, P. P. Chakrabarti, and B. B. Bhattacharya, "Factorization Based Dilution of Biochemical Fluids with Micro-Electrode-Dot-Array Biochips," in *Proc. of the ASP-DAC*, 2019, pp. 462–467.

[33] Shalu, S. Kumar, A. Singla, S. Roy, K. Chakrabarty, P. P. Chakrabarti, and B. B. Bhattacharya, "Demand-Driven Single- and Multitarget Mixture Preparation using Digital Microfluidic Biochips," *ACM TODAES*, vol. 23, no. 4, pp. 55:1–55:26, 2018.

BTP_Batch2_FInal_Report_tin.pdf

ORIGINALITY REPORT

5%	2%	4%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Fidalgo, Luis M., and Sebastian J. Maerkl. "A software-programmable microfluidic device for automated biology", Lab on a Chip, 2011. Publication	1%
2	www.math.gatech.edu Internet Source	<1%
3	Submitted to Indian Institute of Technology, Bombay Student Paper	<1%
4	Yang Zhao, Krishnendu Chakrabarty. "Design and Testing of Digital Microfluidic Biochips", Springer Nature, 2013 Publication	<1%
5	Zipeng Li, Krishnendu Chakrabarty, Tsung-Yi Ho, Chen-Yi Lee. "Micro-Electrode-Dot-Array Digital Microfluidic Biochips", Springer Nature, 2019 Publication	<1%
6	ddfe.curtin.edu.au Internet Source	<1%