University of Cincinnati

Date: 1/21/2014

I, Madhuri N Gupta, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
Multi-Board Digital Microfluidic Biochip Synthesis with Droplet Crossover Optimization

Student’s name: Madhuri N Gupta

This work and its defense approved by:

Committee chair: Wen Ben Jone, Ph.D.

Committee member: Jason Heikenfeld, Ph.D.

Committee member: Andrew Steckl, Ph.D.
Multi-Board Digital Microfluidic Biochip Synthesis with
Droplet Crossover Optimization

A thesis submitted to the

Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the School of Electronic and Computing Systems
of the College of Engineering and Applied Sciences

January 21, 2014

by

Madhuri Nanjundeshwara Gupta
BE (ECE), Visvesvaraya Technological University, 2010

Thesis Advisor and Committee Chair: Dr. Wen Ben Jone
Abstract

Microfluidic lab-on-a-chip technology is a promising replacement for a lot of the equipment in biomedical and chemical laboratories. Digital microfluidics based on EWOD technology, deals with manipulation of liquid droplets using electro-wetting properties, based on a given assay. Many synthesis methodologies have been proposed but we have found the online synthesis approach the most reasonable.

In this work, we attempt to make DMFBs more scalable by suggesting a multi-DMFB architecture approach. This allows us to run large-scale assays in lesser time. Further, we also discuss a novel method to reduce the droplet routing time by introducing an algorithm to reduce droplet exchanges between DMFBs. We present a series of experiments and discuss the inferences and observations for the same. A reduction of up to 40s in runtime was achieved for an assay of 958 operations.
Acknowledgments

I would like extend my gratitude to my thesis advisor, Dr. Wen-Ben Jone, whose constant support and guidance has been crucial in the shaping up of this thesis. His positive remarks and jovial nature made our weekly meetings productive and enjoyable. His promptness in replying to emails made this collaboration less of an effort. I would also like to thank Dr. Jason Heikenfeld and Dr. Andrew Steckl for agreeing to be on my thesis committee and for spending time in reviewing my work.

I also thank Dr. Philip Brisk and Mr. Daniel Grissom from UC Riverside, for their support and guidance with regard to their flagship tool (and my base source code), “MFSimStatic”, their generosity in sharing ideas and viewpoints and for answering my many random questions. I would also like to thank my fellow lab-mate and friend, Mr. Rissen Joseph whose critical analysis and suggestions were important at various stages of my research work. I also thank Dr. Xingguo Xiong of University of Bridgeport for his guidance.

I dedicate this work to my parents and my sister, without their unconditional love, support and encouragement; I would not have come so far. Courses and lab work would have been impossible without my friends Sudarshan and Nakul, who sat through nights debugging code with me. I also thank my friends Ujwal, Kaushik, Bharat, Kumar, Nikhil and roommates, Ranjani, Sambhavi, Subashini and Asawari, who made my life in Cincinnati a wonderful time to reminisce.
Contents

1 Introduction 1

1.1 Continuous-flow microfluidics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Digital (Droplet-based) microfluidics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.2.1 DEP and EWOD methods of droplet control . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 EWOD working principle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.4 Advantages of digital microfluidic chips [1] . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.5 Microfluidic biochip challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
1.6 Thesis focus and organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  1.6.1 Thesis focus and results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  1.6.2 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2 Background 10

2.1 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.1.1 Scheduling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.1.2 Placement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.1.3 Routing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.2 Online versus offline synthesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.3 Fast Online Synthesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.3.1 Topology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.3.2 List scheduling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.3.3 Left-edge binding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
## CONTENTS

2.3.4 Simplified Roy’s maze routing ........................................... 16
2.3.5 Motivation ................................................................. 16

2.4 DMFB Static Synthesis Simulator (SSS) .................................... 17
2.5 Fluidic constraints ............................................................. 19

3 Multi-Board Arch. & its Motivation ........................................... 21

3.1 Motivation ................................................................. 21
  3.1.1 Very large scale assay applications .................................... 21
  3.1.2 Manufacturing Challenges ............................................. 22
  3.1.3 Wiring Challenges and Pin Constrained Designs ...................... 22
  3.1.4 More IO Ports .......................................................... 23
  3.1.5 Test Duration and Complexity ........................................ 23

3.2 Multiple Board Arch. ....................................................... 23
  3.2.1 High-Level Design ..................................................... 23
  3.2.2 Connector Design ...................................................... 24
  3.2.3 Design Constraints .................................................... 25

4 Synthesis & Optimization ..................................................... 29

4.1 Updated synthesis framework ............................................. 30
  4.1.1 Scheduling ............................................................. 30
  4.1.2 Binding ................................................................. 30
  4.1.3 Routing ............................................................... 31

4.2 Integration of partition phase .............................................. 33

4.3 Graph Partitioning .......................................................... 35

4.4 Graph Modeling ............................................................. 35

4.5 Kernighan-Lin Algorithm ................................................... 38
  4.5.2 Bi-partitioning using the KL Algorithm [2] ......................... 39
  4.5.3 Definitions ............................................................. 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.4</td>
<td>Complexity analysis</td>
<td>41</td>
</tr>
<tr>
<td>4.5.5</td>
<td>Post processing steps</td>
<td>43</td>
</tr>
<tr>
<td>4.5.6</td>
<td>Case study</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>Experiments and Results</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Benchmarks</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Graphs and Notation</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>Experiments</td>
<td>51</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Experiment 1</td>
<td>51</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Experiment 2</td>
<td>52</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Experiment 3</td>
<td>52</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Experiment 4</td>
<td>54</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Experiment 5</td>
<td>55</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Experiment 6</td>
<td>56</td>
</tr>
<tr>
<td>5.4</td>
<td>Result interpretation</td>
<td>58</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions &amp; Future Research</td>
<td>59</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Continuous-flow microfluidic chip ............................................. 2
1.2 DEP based array ................................................................. 3
1.3 EWOD based DMFB .......................................................... 4
1.4 Concept of EWOD ............................................................. 5
1.5 Degrees of wetting .............................................................. 5
1.6 (a) EWOD based 2D array of electrodes (b) Cross section view of DMFB .......... 6
1.7 Lab setup schematic diagram ..................................................... 8
2.1 Basic DMFB synthesis with scheduling, placement and routing ..................... 10
2.2 Online and offline synthesis comparison ....................................... 12
2.3 Virtual topology on DMFB ..................................................... 13
2.4 Intra-module routing of droplets for basic operations performed on a DMFB ........ 14
2.5 Left-edge algorithm binding operations to modules ................................ 15
2.6 DMFB visualizer GUI ............................................................ 18
2.7 3D placement view ............................................................. 18
2.8 Compact droplet routing representation ....................................... 19
2.9 Depicts rule 1 of fluidic constraints ......................................... 20
2.10 Depicts rule 3 of fluidic constraints ........................................ 20
3.1 High-level positions for DMFBs ............................................... 24
3.2 Interconnects between DMFBs ................................................. 24
3.3 Electrical layout of a 3-phase bus ............................................. 25
LIST OF FIGURES

3.4 Explosive view of the three layer construction ................................. 26
3.5 Valid multi-DMFB configurations ................................................. 26
3.6 Invalid multi-DMFB configurations ............................................... 26
3.7 Interface between the connector and Flexible modules ...................... 28
4.1 Original synthesis flow ............................................................... 29
4.2 4 DMFB architecture and equivalent routing area ............................. 32
4.3 Assay timeline showing fixed time steps (TS) interleaved with variable length routing phase (R) .......................................................... 32
4.4 New synthesis flow ...................................................................... 34
4.5 Visual partition of PCR DAG ......................................................... 34
4.6 Placed PCR assay ........................................................................ 36
4.7 Graph modelling example for PCR ................................................ 37
4.8 Initial and optimal partitions .......................................................... 40
4.9 1 split Protein DAG .................................................................... 44
5.1 PCR DAG .................................................................................... 50
5.2 In-vitro DAG ............................................................................... 50
5.3 Generic protein DAG ................................................................... 50
5.4 Overhead for protein assays : SPPR .............................................. 53
5.5 Overhead for protein assays : SPR ................................................ 54
5.6 Number of DMFBs ...................................................................... 55
5.7 Connector length sweep for 2_15_7 architecture ............................... 56
5.8 Connector length sweep for 4_15_7 architecture ............................... 57
5.9 Multiples of ProteinSplit_3 DAG .................................................. 57
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Online versus offline synthesis results for protein assay</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Post-placement <em>Node:Operation</em> binding</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>Connectivity matrix</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Synthesis execution time for 2_15_13 architecture, C = 30</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Schedule time vs routing time for 2_15_13 architecture, C = 30</td>
<td>52</td>
</tr>
</tbody>
</table>
Recent advances in the field of microfluidics have generated tremendous interest in the field of miniature devices for biomedical and chemical applications. These integrated micro-systems are also known interchangeably in literature as bioMEMS, lab-on-a-chip and microfluidic biochips. Microfluidic biochips are the recent day answer to repetitive and laborious laboratory processes. They provide a method for quick and efficient operations and handling of nano-liters (nL) of liquid, which provides for ultra-sensitive detection [3]. All the benefits of miniaturization such as smaller sample requirement, reduced reagent consumption, decreased analysis time and higher levels of throughput and automation are realized through the use of biochips. They help reduce cost per assay as well as laboratory space. They also allow precision in experimentation, e.g., we can uniformly heat a small droplet better than a larger amount of liquid. Microfluidic biochips have a wide array of applications. They are popularly used for enzymatic analysis, DNA analysis (e.g., PCR and nucleic acid sequence analysis), clinical diagnostics, analysis of proteins and peptides, immunoassays, toxicity monitoring [4] [5], point of care devices, biomedical and chemical experiments, continuous sampling and real-time testing of air/water samples for biochemical toxins and pathogens [3].

1.1 Continuous-flow microfluidics

The early biochips were based on continuous flow of liquids through micro-pumps, micro-valves and permanently etched micro-channels fabricated in glass, plastic or other polymers (as shown in Fig. 1.1). Liquid flow was actuated using external pressure sources, integrated mechanical micro-pumps or by electro kinetic
mechanisms such as electro-osmosis \cite{6} \cite{7}. They were designed for simple and specific biochemical assays that do not require complex manipulations of liquid. The liquid flow through the micro-channels is governed by multiple parameters such as pressure, electric field and fluid resistance. The variation in these parameters along the flow path make the fluid flow in any location dependent on the properties of the entire system. There have been some synthesis and automation methods for continuous flow systems such as valve switching minimization techniques \cite{8}, but they are largely non-reconfigurable and fixed structures. Low reconfigurability also implies low fault tolerance. The next generation of biochips called \textit{droplet-based microfluidic} chips will be discussed in the following section.

\section{Digital (Droplet-based) microfluidics}

Analogous to the world of electronics, an alternative solution called \textit{digital microfluidics} was proposed. In this type of system, liquids in the form of droplets are controlled independently and concurrently over a 2-D array of cells (or electrodes). The prime advantages of digital microfluidic chips are reconfigurability, ease of integration and ability to scale. The assays can be broken down to a few basic operations that can be easily defined and automated. Droplet-based protocols are functionally equivalent to traditional bench-scale chemistry and assays can be scaled down, automated and integrated. A group of cells can be
reconfigured to perform different activities at different phases of the experiment. Due to these properties, we can consider biochips to be *programmable microfluidic processors*. Droplets can be manipulated using chemical, thermal, acoustical and electrical principles. Manipulation of droplets can happen through thermo-capillary transport [10], surface acoustic wave transport [11], and electrical methods such as dielectrophoresis (DEP) [12] [13] (see Fig. 1.2) and electro wetting (EWOD) [14] [15] (see Fig. 1.3). As the complexity of these devices increases and larger and more assays need to be executed concurrently on-chip, we can no longer rely upon full custom designs. Computer Aided Design (CAD) based tools become essential as it did in the VLSI industry. CAD tools help the biochip users to concentrate on the high level assay and leave the details of implementation to the tool. They help reduce human effort and enable high volume production of biochips.

### 1.2.1 DEP and EWOD methods of droplet control

Liquid DEP actuation is defined as the attraction of polarizable liquid masses into the regions of higher electric-field intensity. DEP relies on high frequency AC voltages ($230 - 300 V_{rms}$ at $50 - 200 kHz$ [16]). DEP is based on electrohydrodynamic forces and provides high droplet speeds. The disadvantage of using DEP is the excessive Joule heating [3]. Fig. 1.2 depicts a 2D DEP-based array of electrodes.
1.3 EWOD working principle

In this section, we shall discuss EWOD chips and their working principle in detail. Electro-wetting-on-dielectric (EWOD) refers to the modulation of the inter-facial tension between a conductive fluid and a solid electrode coated with a dielectric layer by applying an electric field between them \[3\]. Unlike DEP, the EWOD method uses DC (or low frequency AC) to directly control the inter-facial energy between a solid and a liquid phase. The dielectric layer blocks the DC electric current. As a result, aqueous solution with salt concentration of 0.15 M can be actuated without Joule heating \[18\]. Fig. 1.4 (top) depicts the hydrophobic nature of the liquid droplet when there is no voltage applied. However, on application of a voltage, we see that the liquid turns hydrophilic (Fig. 1.4 (bottom)). Application of voltage between the droplet and the electrode changes the distribution of electric charge in the droplet and significantly decreases the contact angle. Fig. 1.5 depicts a liquid drop on an insulated surface with a high contact angle $\theta_0$. Here, $\theta$ represents the smaller angle of contact during electro-wetting. The polarity of voltage in Fig. 1.4 is arbitrary, and electro-wetting will occur in both directions \[19\].

Fig. 1.6 depicts two parallel glass plates. The lower plate is the active plate which is patterned with a 2-D array or electrodes, and each of which is provided a separate pin connected to a DC source. The top plate is a common ground plate coated with a continuous ground electrode. A dielectric insulator (i.e. parylene-C coated with a teflon-AF) is added to the top and bottom plates to reduce the wettability of the plates and to add capacitance between the control electrode and the droplet. A filler medium such as silicone oil is sandwiched between the two plates. Droplets can move between the plates in the silicone oil. When
Figure 1.4: concept of EWOD [19]

Figure 1.5: Degrees of wetting [19]
a control voltage is applied, the droplet is hydrophilic; when it is not applied, it becomes hydrophobic (see Fig. 1.5). If the electric field is applied to only one side of the droplet, it creates an imbalance of interfacial tension which forces the droplet to move. Thus, in order to move a droplet to an adjacent electrode, a control voltage is applied to the neighboring electrode and the voltage being applied to the current electrode is deactivated. Droplet size must be maintained large enough to cover the electrode and overlap slightly onto neighboring electrodes in order to achieve droplet movement. By varying the electrical potential along a linear array of electrodes, electro-wetting can be used to move nano-liter volume liquid droplets along this line of electrodes [21]. The velocity of the droplet can be controlled by adjusting the control voltage (0-90 V), and droplets can be moved at speeds of up to 20 cm/sec [22]. Droplets can be transported in user defined patterns and under clocked-voltage control over a 2-D array of electrodes.

1.4 Advantages of digital microfluidic chips [1]

- Unlike continuous flow devices, digital microfluidic chips are under software-driven electronic control. A typical setup (see Fig. 1.7) would include a computing device such as a laptop or even a mobile device connected through a USB interface with a PCB. We can mount our microfluidic device on the PCB along with a micro-controller and other components that can process the activation patterns and drive the electrodes.

- Assays can be designed as a graph made from an elemental set of operations such as merge, mix, split, transport, detect etc. Each of these operations can be automated and therefore, the assay can be
1.4. ADVANTAGES OF DIGITAL MICROFLUIDIC CHIPS

- All operations are carried out between the two parallel plates, and there is no need for moving parts such as micro-pumps and micro-valves.
- Controls or prevents evaporation with the oil surrounding the droplets.
- There are no permanent channels. Channels exist in the virtual sense and can be reconfigured through software.
- Many droplets can be individually controlled since all the electrodes are individually actuated. This helps in running multiple operations in parallel and helps complete complex assays quickly.
- There are no ohmic currents. Capacitive currents exist but DC is blocked, and Joule heating and electrochemical reactions do not occur.
- Works with a wide variety of liquids and most electrolyte solutions will work.
- There is very efficient sample utilization and almost no wastage of liquids. This can be very important in case of expensive fluids or forensic samples.
- They are energy efficient and can be used in point of care devices.
- High droplet transport speeds of up to 20 cm/s can be achieved.
- Conditional execution can be programmed through feedback to control the next step based on outcome.
1.5 Microfluidic biochip challenges

Digital microfluidic biochips are fabricated using standard micro-fabrication techniques. Due to the underlying mixed technology and multiple energy domains, they exhibit unique failure mechanisms and defects. Testability is a major concern in microfluidic chips \cite{23,24}. They are used in medical diagnosis where thorough testing is critical. Many defects can be present in the device at once. Diagnosing the presence of a fault is the primary challenge. Several CAD methods have been proposed to get around this issue by reconfiguring the chip to avoid faulty regions of the chip \cite{24,26}. Reconfiguration can be used to re-map the operations that were originally mapped to the faulty locations. This helps in increasing the yield since we have a way to make use of faulty chips as well. Most CAD tools that are available are MEMS tools that can be used for simulating physical attributes of the device and as characterization tools. They allow analysis of the fluidic and physical properties and represent a bottom-up design approach. We concentrate on the top-down approach for the synthesis process. An efficient design methodology and framework is required and the one provided by UC Riverside \cite{27,28} is a good place to start for development of better algorithms for synthesis or for new architecture design.
1.6 Thesis focus and organization

1.6.1 Thesis focus and results

The focus of this thesis is two-fold. The existing synthesis framework [27] and associated algorithms are all based on a single DMFB. The main aim of this thesis is to extend this framework to be able to handle one or more DMFBs at once and be able to use them to solve bigger assays which would be too big to run on a single chip. We found that, as the assay complexity increases, the ratio of routing time to scheduled operation time increases as well. Hence, the second aim of this thesis is to reduce the time spent on routing droplets between DMFBs by reducing the frequency of droplet exchanges. This is especially important for long connectors as we have shown in our chapter on experimental results. Increase in connector length causes linear increase in routing time and it has more impact for larger assays. Our results also show that, by adding more DMFBs, bigger assays that fail to synthesize on smaller setups can be run in reasonable time. Increasing the complexity of the assay (levels of splitting) causes exponential increase in routing time, whereas increasing the number of copies of the same assay causes linear increase in routing time.

1.6.2 Organization

- The second chapter provides the background for understanding the thesis better. We discuss the single DMFB synthesis and associated algorithms, different schools of thought such as offline and online synthesis methods and their trade-offs, the primary operations and how they translate into droplet movement and fluidic constraints.

- The third chapter discusses the motivation for multi-DMFB approach in detail. It also discusses high-level design and connector design for the new multi-DMFB architecture and the design constraints.

- The fourth chapter discusses the major changes in the existing phases of synthesis. It also throws light on an optimal partitioning algorithm and its motivation along with a case-study.

- Chapter 5 discusses the experimental setup and experiment design along with results and discussion.

- Chapter 6 provides the conclusion and future work.
Chapter 2

Background

This chapter deals with some of the topics that would serve as a background, required to fully understand and appreciate the later chapters. We discuss several different synthesis methods and describe the basis for our choice.

Microfluidic biochip synthesis is a top-down design methodology that converts a behavioral description of an assay into a programmable set of well-defined basic operations. Fig. 2.1 shows progression of the synthesis process and realization of the assay from a DAG to a more realistic electrode activation sequence generated from the output of a router. We discuss different schools of thought in solving the synthesis problem. Note that we are only considering direct-addressing DMFBs which have one control pin for each electrode.

Figure 2.1: Basic DMFB synthesis with scheduling, placement and routing [29]
2.1 Related work

2.1.1 Scheduling

Scheduling methods have been extensively studied in computer science. Operating systems schedule different processes and events on the computer, based on limited resource availability. The objective of a scheduler is to find the start and stop times of each operation such that the overall run time is minimized. Scheduling is known to be an NP-complete problem, and therefore we need to develop heuristics for scalability.

Researchers at Duke University have performed a lot of research in trying to solve the synthesis problem along with fault tolerance approaches over the past decade and more. They have proposed offline synthesis methods such as integer linear programming (ILP) [30], modified list scheduling (MLS) and genetic algorithm (GA) methods to solve the scheduling problem. These schedulers assume the entire DMFB to be a uniform 2D array of cells. They consider virtual resources of sizes 2x2, 2x3 and 2x4 and each size results in a different operation time which is empirically determined. It is a complex problem since a 2D array can be divided into modules of different sizes in numerous ways. Operation run times depend on resource sizes. At each time step, we have a new layout and this results in very high number of possible solutions. ILP algorithm solves the scheduling problem using a mathematical method and provides an optimal result, but it takes a long time to compute the result and is not scalable. GA is an iterative algorithm which computes near optimal to optimal results. However, it is takes a long time to compute the schedule and is prohibitive to use in point of care devices. Other scheduling algorithms such as Ricketts’ hybrid genetic algorithm [31] and Maftei’s tabu search scheduler [32] are iterative improvement algorithms which spend anywhere from 4 seconds to 1 hour to compute schedules.

2.1.2 Placement

All electrodes on the DMFB are capable of performing basic operations such as mixing, splitting, merging, transport and storage. Hence, these basic operations can be performed anywhere on the DMFB. Placer’s main objective is to pack as many concurrent operations as possible in a given area. Several direct-addressing placement and unified scheduling-placement algorithms [24],[26] that use simulated annealing have been proposed. They run in minutes or tens of seconds [29].
2.1.3 Routing

Su et al. route droplets in a sequential order and they redo the placement when routing fails [25]. BioRoute [33] is an algorithm that uses max-flow min-cost to compute many routes at once followed by a negotiation-based detailed routing. There are other methods as well [15, 34] but most of these are designed for offline routing and run times have not been discussed. Both, BioRoute [33] and Huang’s algorithm [34], report run times below 1s on a desktop PC [29].

2.2 Online versus offline synthesis

The discussion above clearly shows that most methods are not suitable for online computation which is essential for real-time response to errors and control. The work by Grissom and Brisk [29] discusses heuristics and certain topology constraints that make the synthesis process much simpler and drastically improves computation times at all levels of synthesis. Their work discusses a less optimal but faster approach to deal with this issue thereby providing the essential groundwork for the multi-DMFB synthesis proposed in this research. Shorter synthesis time is an important feature for point of care devices and fault resistance. It also is more scalable, and hence we consider it as a feasible approach for multi-DMFB architectures. Fig. 2.2 clearly shows the advantages and disadvantages of using faster methods of synthesis that are near optimal. The work in [29] concludes that we can settle for less optimal results in favor of much faster computation time. In the next section we discuss the algorithms used for fast online synthesis in detail. We modify them to suit multiple-DMFB synthesis as discussed in chapter [3].

Figure 2.2: Online and offline synthesis comparison [29]
2.3 Fast Online Synthesis

2.3.1 Topology

The online synthesis design proposed in [29] considers a DMFB with a virtual topology as in Fig. 2.3. The topology fixes the routing and non-routing regions and reduces the ability to reconfigure. In return, we get a much simpler and hence, faster computation. Once the virtual module is placed on the DMFB, the ones placed over a physical detector/heater will be considered under special modules. Fig. 2.4 from [29] depicts all the basic operations and entry-exit points. The fixed entry/exit to/from modules simplifies and standardizes the routing and prevents deadlocks. Basic operations such as mixing and splitting are fixed in their action and, by stitching them up according to the assay, we can generate quick results.

2.3.2 List scheduling

List scheduling algorithm is a greedy and constructive algorithm that computes the schedule faster than other iterative or randomized algorithms. Each operation (node) in the assay (DAG) is scheduled exactly once. List scheduling maintains an array called availMods[] that holds the number of modules for each type.

![Virtual topology on DMFB](29)

Figure 2.3: Virtual topology on DMFB [29]
Figure 2.4: Intra-module routing of droplets for basic operations performed on a DMFB [29]
Figure 2.5: Left-edge algorithm binding operations to modules [29]

(basic, detect, heat) that is present on the DMFB, as a result of the virtual topology. The algorithm assigns a priority value called *critical path priority* (CPP) to each node in the DAG. CPP for any node \( n \) is the largest summation of time steps from (and inclusive of) \( n \) to any output in its fan-out. LS starts out with a candidate list that consists of all the nodes that have *dispense* nodes as the parent. The main loop continues until every node is scheduled. In each iteration, the scheduler tries to schedule as many operations as possible with the available resources. In order to reduce the overall time steps, the scheduler orders the nodes in decreasing CPP value and schedules the node with the highest CPP first. As discussed before, \( \text{availMods}[\ ] \) holds the number of free resources at each time step. At the start of each operation, we decrement the resource count by one and increment it at the end of that operation. An operation can only be scheduled if its parent operations are all completed. To understand this algorithm in detail, refer to the pseudocode in [29].

### 2.3.3 Left-edge binding

Placement is an NP complete problem [26]. By creating the virtual topology for a DMFB, reconfigurability of the DMFB is reduced, but it also reduces the complexity of placement to a binding algorithm.

The work in [29] explains the left edge binding algorithm which is a popular algorithm used in channel
routing. It has a complexity of $O(|V|^2)$ where $|V|$ is the number of assay operations. A fixed-module bin is created for each module in the virtual topology. All the nodes are sorted into operation-bins based on module type. The binder sorts the nodes in each operation-bin in order of start time. In Fig. 2.5 we can see that there are two basic modules, one heat module and one detect module. The first module, ModBin1, finds the operation-bin that matches its type (OpBin1). The binder then adds the operations from OpBin1 to ModBin1 as long as the start time of the operation being added currently does not conflict with other operations already present in ModBin1. That means, none of the operations in ModBin1 should have a stop time later than the current operation’s start time. If the operation has been added to ModBin1, it is removed from OpBin1. Once the binder has gone through all the operations in OpBin1, it moves on to do the same process for ModBin2 and so on. This results in a set of operations being bound to a physical entity (module).

2.3.4 Simplified Roy’s maze routing

The last step is to route the droplets between modules at the end of an operation, if the next operation is scheduled in a different module. Routing is also required between IO ports and modules. A simplified version of Roy’s maze router [35] has been used in [29]. It uses Soukup’s fast maze router [36] to compute quick sequential routes for droplets and then compacts them together by adding stalls in the middle to avoid any droplet interference [29]. A route is a collection of routing points that define the droplets path from its source (dispenser or module) to its destination (module or output). Each droplet has an associated route and a collection of these routes is called a routing sub-problem. The router computes a routing sub-problem at the end of every time step and before the beginning of the next time step. At this stage, the routes in the sub-problem are not compacted. Interferences are not resolved and every droplet has been routed independently. In the compaction phase, each droplet’s route is checked with all others to see if there is any interference and stalls are added at the beginning (or midway) to avoid the same. Since the virtual topology places modules with a fixed entry and exit points, there is no issue of deadlock. For a more detailed discussion, refer to [29].

2.3.5 Motivation

From the experimental results presented in [29], we discuss some of them in Table 2.1. We can see that online flow = (list scheduler + left edge binder + simplified Roy’s router) provides a very robust, quick and
near optimal result. The offline flow = (genetic algorithm + SA placer + Roy’s router) provides the better assay time (AT) but a significantly higher computation time (CT). This is an important metric that prompted us to use the online synthesis approach which is more scalable since it is crucial for multi-DMFB synthesis approach.

Table 2.1: Online versus offline synthesis results for protein assay

<table>
<thead>
<tr>
<th>Protein assay</th>
<th>schedule AT(s)</th>
<th>placement CT(s)</th>
<th>routing AT(s)</th>
<th>routing CT(s)</th>
<th>Total time AT+CT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline synthesis</td>
<td>110</td>
<td>22</td>
<td>79532</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Online synthesis</td>
<td>116</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

2.4 DMFB Static Synthesis Simulator (SSS)

This section describes the synthesis framework as made available by UC Riverside [28]. The SSS tool can be used to simulate and visualize the various phases of the synthesis process. It has been designed for modularity and the users can implement their own individual modules (schedulers, placers, routers) and integrate them into the system. Users can run individual modules or the whole flow at once. The system uses text-based I/O interfaces to share information between different phases. It can serve as a great comparison tool where one can compare the performance of different algorithms on a common platform.

The framework has a C++ back-end which includes file-I/O, interface standards, timing mechanism etc built in. This helps in fair comparison of algorithms. Developers need to change only a few functions to include their module in the framework. For the end-user who would like to use the tool for non-developmental purposes, a simple Java based GUI (see Fig. 2.6) has been provided. It provides a graphical output that is great for debugging and presentation purposes. The GUI reads the text output files generated by the C++ simulator and converts them to useful visuals, videos and 3D renderings. The visualizer provides outputs of the input DAG, scheduled DAG, placed DAG, 2D placement, 3D placement (see Fig. 2.7) which is a view of the placement of operations at each time step. It also provides a compact routing (as in Fig. 2.8) and detail routing output in the form of a series of images. We can also create a video of the droplet movement. For more information including a step-by-step video on the initial setup etc., refer [28].
2.4. DMFB STATIC SYNTHESIS SIMULATOR (SSS)

Figure 2.6: DMFB visualizer GUI

Figure 2.7: 3D placement view, horizontal places represent time steps
2.5 Fluidic constraints

We discuss the fluidic constraints as defined in [25]. During droplet routing, we need to maintain a minimum distance between any two droplets to prevent them from merging accidentally. Let us consider two droplets $D_i$ and $D_j$. First, to avoid mixing, $D_i$ and $D_j$ cannot be in adjacent or diagonally adjacent electrodes at the initial time slot $t$. Let us define the electrode array as $(X, Y)$ and let $X_i(t)$ and $Y_i(t)$ define the position of $D_i$ at time $t$.

\[
|X_i(t) - X_j(t)| \geq 2 \\
|Y_i(t) - Y_j(t)| \geq 2
\]

At the next time step $t + 1$ the following rules must hold:

**Static rule:**

**Rule 1:** $|X_i(t + 1) - X_j(t + 1)| \geq 2$ and $|Y_i(t + 1) - Y_j(t + 1)| \geq 2$

This implies that the next locations for the two droplets must not be adjacent either.
2.5. FLUIDIC CONSTRAINTS

Figure 2.9: Depicts rule 1 of fluidic constraints [25]

Figure 2.10: Depicts rule 3 of fluidic constraints [25]

Dynamic rules:

**Rule 2:** $|X_i(t+1) - X_j(t)| \geq 2$ and $|Y_i(t+1) - Y_j(t)| \geq 2$

This implies that the activated cell for $D_i$ cannot be adjacent to $D_j$. Otherwise, $D_j$ will have more than one active adjacent cells and can cause erroneous droplet movement or splitting.

**Rule 3:** $|X_i(t) - X_j(t+1)| \geq 2$ and $|Y_i(t) - Y_j(t+1)| \geq 2$
Chapter 3

Multi-Board Architecture and its Motivation

In the previous chapters, we have described the basic concept of EWOD-based digital microfluidic biochips (DMFBs) and their synthesis process. In this chapter, we propose the idea of multi-DMFB design with the objective of:

- Reducing the execution time of large scaled assay operations
- Allowing extremely large scaled operations that cannot be executed by a single DMFB to be executed using multiple DMFBs.

3.1 Motivation

3.1.1 Very large scale assay applications

Multi-DMFB architectures allow us to run very large assay applications in reasonable time. They may also allow us to run those assays which are considered too big to be executed on currently available DMFBs. Reducing execution time can greatly increase throughput and make more assays feasible.
3.1.2 Manufacturing Challenges

DMFBs are in their nascent stage when it comes to high volume manufacturing for commercial use. In order to run multiple concurrent experiments, we need more inputs and larger area for the assays to be executed in parallel. Unfortunately, it is difficult to manufacture large boards which are defect-free. Larger area and density of DMFBs will reduce the yield, especially for new technology nodes, and this deters high volume manufacturing [37].

3.1.3 Wiring Challenges and Pin Constrained Designs

One of the foremost challenges in developing a feasible design of a DMFB is the wiring requirement. In the direct addressing scheme, each electrode in the array is independently accessible via a control pin. For example, a small or medium sized (<10x10) DMFB with individually addressable electrodes requires 100 pins to be routed to the periphery of the board for electrical connections. Therefore, we have a quadratic relationship between the board size (or edge length) and the pin count. This puts an upper bound on the size of the DMFB due to boundary limitations and interconnection complexity [35]. A large-sized DMFB (>100x100 electrodes) will need a complicated multilayer routing solution which is not reliable and is expensive [38]. There has been prior research with regard to pin-constrained designs and proposals such as Broadcast Electrode Addressing [39], Droplet-Trace Based Array Partitioning Method [40], Cross-Referencing Based Method [41], N-Phase or Bus-Phase Addressing [5], ILP-Based Pin-Count Aware Design Method [42] have been made. The broadcast addressing method makes the DMFB specific to the assay, since the pins are shared based on the droplet positions, which are assay-specific. The Droplet-Trace based array partitioning method also makes the DMFB more assay-specific. Cross referencing is a method of using \((M+N)\) pins to control \((M \times N)\) electrodes and it cannot be used for high throughput assays. These DMFBs cannot handle multiple droplets simultaneously due to electrode interferences. N-Phase Addressing method is useful only for a \(1 \times N\) board and hence, is of limited use. This provided a motivation to find a solution that can overcome all or most of the issues discussed above.
3.1.4 More IO Ports

Having multiple DMFBs has an advantage of higher edge length for the same area of operation. This provides an advantage in certain assays that are IO-intensive. Let us illustrate this with an example. If we make a single DMFB of size \((M \times N) = K\) cells, we have \((M + N)\) cells along the edges where we can have the IO ports. This is under the assumption that we have IO ports at every alternate cell on the edge of the DMFB to satisfy the fluidic constraints as described in [2]. Instead, if we use two smaller DMFBs of size \(m = M/2\) and \(n = N\) (for a total of \((2*M/2*N) = K\) cells), we get \(2*(m+n) = (M + 2N)\) IO ports. We would then get \(N\) extra IO ports (minus any connectors needed between the DMFBs). More details regarding connectors will be provided in the latter sections.

3.1.5 Test Duration and Complexity

Testing DMFBs is crucial due to their application in safety-critical biomedical applications. There are several papers that describe the testing methodologies in DMFBs which include structural and functional testing. The faults can be catastrophic (hard) or parametric (soft). Testing can be done both, online and offline. A fault uncovered during any of the above processes will term the DMFB as faulty. The test time is far lesser for small DMFBs and can potentially be done in parallel on-chip [43] and on multiple DMFBs.

3.2 Multiple Board Architecture

3.2.1 High-Level Design

In this section, we will discuss the design and constraints of multiple board architecture. We consider a grid-based design for the positions of the DMFBs. Every DMFB in our design will hold a pair of X and Y co-ordinate values that describe its position in the grid and provide us with the relative location of the DMFB with respect to the others (Fig. 3.1). We can have any number of DMFBs in our architecture but are subject to some limitations as discussed later.

The DMFBs need to be connected so that droplets can be transported across from one DMFB to the other. We discuss these connectors in the next section.
3.2.2 Connector Design

Adding to the design discussed in the previous section, we now include connectors in the design. These are required for droplets to be transported from one DMFB to the other. Connectors run north-south or east-west as shown in Fig. 3.2. We can have multiple connectors between any two DMFBs but for simplicity we consider a single connector between two DMFBs.

Similar to microfluidic chips, droplet transfer can be *continuous* or *discrete*.

*Continuous Flow:*

The most common method of liquid transfer is by using miniature teflon tubes bonded onto the module using
epoxy glue. The liquid is pumped through the tube using a micro-pump and a micro-sensing equipment may be used [44]. Multiple capillary tube based \textit{ribbon} like interconnects have been fabricated [45]. These methods are well suited for continuous-flow based chips but not so much for DMFBs.

\textit{Discrete Flow:}

An alternative to continuous flow is transfer of liquid as droplets. Since DMFBs process the liquids in droplet form, the same can be extended to form connecting devices as well. A $1 \times N$ array of electrodes can be used for two-way transport of droplets. They can be individually addressable or can be implemented using a 3-phase bus design [46] (see Fig. 3.3). The design in [46] proposes a flexible interconnect which can support DMFB stacking as well. Figs. 3.4 and 3.7 pictorially depict the construction of a flexible module and a connector module that interfaces with the DMFB.

\section*{3.2.3 Design Constraints}

For ease of implementation, we have considered the following constraints on the design.

1. All the DMFBs are of the same size (both in length and width) and are used in the same orientation.

2. All connectors are of the same length.

3. We have only one bridge between any two DMFBs and they must be vertical (north-south) or horizontal (east-west).
CHAPTER 3. MULTI-BOARD ARCH. & ITS MOTIVATION

3.2. MULTIPLE BOARD ARCH.

Figure 3.4: Explosive view of the three layer construction

Figure 3.5: Valid multi-DMFB configurations

Figure 3.6: Invalid multi-DMFB configurations
4. All our DMFBs have individually addressable pins. We can further extend to include pin-constrained DMFBs as well.

5. Connector can be positioned at any point along the edge of the DMFB keeping the fluidic constraints in check.

6. All DMFBs have to be connected by at least one connector.

7. DMFBs must align to their global position grid space as described in Fig. 3.1. In Fig. 3.5, options ‘a’, ‘b’ and ‘c’ are all valid as they satisfy all the constraints as specified in 3.2.3. In Fig. 3.6, option ‘d’ violates constraint 7 and 2. Option ‘e’ violates constraints 2 and 3. Option ‘f’ violates constraint 6.
Figure 3.7: Interface between the connector and Flexible modules
Chapter 4

Synthesis with Partition Optimization

Motivated by the need for multiple DMFBs to perform large experiments, the task remains on how to run the assay on multiple DMFBs. In this chapter, we discuss the major updates in the synthesis framework, the idea of partitioning, and the algorithm used.

Figure 4.1: Original synthesis flow
4.1 Updated synthesis framework

Fig 4.1 shows the original synthesis flow which involves scheduling, placement and routing. In the next section, we briefly discuss all the changes we made to the synthesis flow to incorporate multiple DMFBs.

4.1.1 Scheduling

The scheduling algorithm that we considered is based on the list scheduling (LS) algorithm. It is one of the fastest scheduling algorithms that has been implemented, and gives near-optimal results. Thus, it is suitable for online synthesis. LS is a greedy, constructive algorithm in which each operation (node) in the assay (DAG) is scheduled exactly once \(^{29}\). The following steps describe the procedure:

- A virtual topology is placed on each DMFB, and the number of special modules is calculated, based on the positions of external devices, for each DMFB. The rest of the modules are considered basic modules.
- An array called \text{AvailMods} / stores the number of modules for each type.
- We aggregate these modules by type for each DMFB and store them in a common array denoted by \text{TotalAvailRes} /.
- We allow LS to take \text{TotalAvailRes} / as the input.

In this manner, LS considers all the DMFBs to be available and generates a schedule based on the aggregated resource count. Note that there was no change to the basic LS algorithm. More information on the algorithm and implementation can be found in \([29]\).

4.1.2 Binding

Placement of operations is done using a binder based on the left-edge algorithm. Operations are bound to pre-assigned modules in accordance with the schedule that has been pre-computed. The inputs to the binding algorithm are the scheduled DAG and the available resources. Binding is handled in a way similar to scheduling. We consider modules from all DMFBs together and provide them to the binder which binds
the operations to the modules based on type, in the order of priority (earlier start time). Hence, there is no change to the binding algorithm except that we provide TotalAvailRes[] instead of AvailRes[] to the binder.

4.1.3 Routing

The routing algorithm we have used is Roy’s maze router. It is based on the Soukup’s fast maze router\cite{36} which produces sequential routes. The result is then compacted by adding stalls to prevent interference. One of the inputs to the router is the scheduled and placed DAG. The second input is a modified Soukup’s board which is a data structure made up of a 2-D array of cells that is used for routing. It is termed modified since it incorporates multiple DMFBs instead of just one. In Fig. 4.2(a), there are four DMFBs arranged in a 2x2 format, with connectors as shown. In this fixed topology design, droplets can reach any point during the routing phase. We model the four-DMFB design on a single board by marking out areas (in gray) apart from the DMFBs and connectors as shown in Fig. 4.2(b). This can be achieved by using the position information of each DMFB which provides the relative position of the DMFB in the global space as shown in Fig. 3.1. We also require connector orientation, position and length information. The routing process takes place as follows:

- Create the multi-DMFB Soukup-cell board with regions marked out as shown in Fig. 4.2(b).
- Create more blockages for modules that are currently being used (for an operation or for droplet storage).
- For each droplet, mark source and target and unblock the region around it.
- By using Soukup’s algorithm, find the shortest path from source to target without considering other droplets’ relative position in that time step.
- Compact all the droplet routes by adding stalls (at the beginning or in the middle) to prevent interference.

In this manner, we achieve routing for multiple DMFBs without changing the core routing algorithm but by providing a complex arena to route in. This completes the synthesis process for multi-DMFB architectures.
CHAPTER 4. SYNTHESIS & OPTIMIZATION

4.1. UPDATED SYNTHESIS FRAMEWORK

Figure 4.2: 4 DMFB architecture and equivalent routing area [29]

Figure 4.3: Assay time line showing fixed time steps (TS) interleaved with variable length routing phase (R) [29]
CHAPTER 4. SYNTHESIS & OPTIMIZATION  4.2. INTEGRATION OF PARTITION PHASE

We solve the routing problem in multiple phases (in sub-problems). Routing time is not accounted for in the schedule and is a variable-time phase between every time-step (TS) of scheduled operations (Fig. 4.3). In [29], we see that routing time, especially for larger DMFBs and bigger assays, is not a small value that we can ignore. This is especially true for multi-DMFB architectures. In order to further speed up the routing process, we can add a new stage in the synthesis process known as partitioning optimization, which we discuss in detail in the following sections of this chapter.

4.2 Integration of partition phase

The Input assay file is a DAG where vertices model operations such as mixing, splitting etc. and edges model dependency in time. The DAG can be split into $k$ sections and we can run the assay on $k$ DMFBs instead of just one. For example, see Fig. 4.5 where there is a marked partition of activities between two DMFBs (green and red) for the PCR assay. This is implicitly done by the schedule-place-route scheme as discussed in section 4.1. We would like to discuss about how we can optimally partition the DAG so that we reduce runtime and make optimal use of all the available resources.

With the unscheduled DAG, we lack information on the final schedule and the eventual location of each operation. The challenge here is to identify the parameter we would like to optimize during partitioning. Since we have a $1 \times N$ connector between each pair of DMFBs, it becomes a bottleneck for routing of droplets since fluidic constraints need to be maintained at all times. Droplets need to maintain 1-cell distance at all times. This can increase the execution time as compared to running it on a single DMFB. Therefore, we can model the droplet crossover frequency as the cost function to be minimized. After the placement stage, we have the required information to derive the droplet crossover frequencies since operations are now bound to specific work modules / IO ports. Hence, we insert a new phase called partitioning between the placement and routing stages as shown in Fig. 4.4.
CHAPTER 4. SYNTHESIS & OPTIMIZATION

4.2. INTEGRATION OF PARTITION PHASE

Figure 4.4: New synthesis flow

Figure 4.5: Visual partition of PCR DAG
4.3 Graph Partitioning

We define graph partitioning as a problem of partitioning nodes of a graph \( G(V, E) \) with cost on its edges into subsets of given sizes, so as to minimize the sum of costs on all edges cut (cutset) \[47\]. Here \( V \) represents the set of nodes or vertices and \( E \) is the set of edges. It is a well-known NP complete problem and we have a number of heuristics that solve the problem in polynomial time. We consider the K-way Kernighan-Lin (KL) partitioning algorithm described in \[47\] for its simplicity, generality and well defined procedural approach. There may be more advanced algorithms with better performance but the KL algorithm is more than sufficient for the problem size that we deal with. The basic KL algorithm is essentially a bi-partitioning algorithm that yields two equal-sized subsets with a small cut-set. By doing it \( \binom{k}{2} \) times (for each pair of DMFBs), we can expect to reduce the overall cut-set. Note that we considered other algorithms such as the max-flow min-cut algorithm and the clustering method. A major drawback for these algorithms is that there is no efficient way to constrain the sizes of the partitions \[47\]. In the clustering method, we get inferior results when we try to constrain the cluster size.

4.4 Graph Modeling

As discussed above, we need the placed DAG as reference to create a graph \( G(V, E) \) on which we perform partitioning optimization. Consider Fig. 4.6 which is a placed version of the PCR assay benchmark. Values in a pair of parentheses represent the id of a DMFB entity that the operation is bound to. Note that the placed DAG will also have schedule information which has been skipped here for illustration purposes.

To understand the process of graph modeling, consider Fig. 4.7 where (a) shows a multi-DMFB setup (\( k = 2 \) in this example). We can see that each DMFB has two work modules for operations such as mix, split, detect etc., and five IO ports for input/output operations. The interference region is created to conform to the fluidic constraints. Fig. 4.7(b) marks each entity with a node number and depicts droplet transfers between nodes according to the placed DAG in Fig. 4.6. Note that nodes 11 and 12 have two droplets transfers between them, as marked in red on the placed DAG.
Some nodes may not be involved in any operation, such as node 4. This is allowed, since we just consider them to have edges of zero weight. Fig. 4.7 (c) Shows the resulting graph $G$ where the nodes are color coded according to type. Work modules can be exchanged with each other if they are of the same type (basic, detect, heat+detect etc.). This is in order to keep the DMFB architecture fixed. In this example, all the work modules are of the same type (basic). IO ports can be dispensers or outputs but are considered the same type. This is under the assumption that all ports can be used interchangeably. Should this assumption be false, we would need to consider them as two separate types (dispense, output) and allow exchanges among like types only. The red dashed line shows the partition with a cut-set of 2. Fig. 4.7 (c) marks the starting point for the partitioning algorithm described in Section 4.5.
Figure 4.7: Graph modelling example for PCR
(a) DMFB pair with marked regions, (b) mapping of DMFB entities to a graph and (c) resultant undirected graph with weighted edges
4.5 Kernighan-Lin Algorithm

Kernighan-Lin (KL) Algorithm is one of the most popular bi-partitioning algorithms. It is an iterative, deterministic heuristic procedure to partition a graph into $k$ (where $k \geq 2$) equal parts. It is characterized by a connectivity matrix $C$. Element $c_{ij}$ represents the sum of weights of the edges connecting elements $i$ and $j$. In our problem, since edges are of unit weight, $c_{ij}$ is simply the count of the number of edges connecting $i$ and $j$. The output of the partitioning method is a pair of sets $A$ and $B$ such that $|A| = n = |B|$, $A \cap B = \emptyset$, such that the size of the cut-set $T$ (shown below) is minimized.

$$T = \sum_{a \in A, b \in B} c_{ab}$$

4.5.1 K-way KL Algorithm: An Overview [2]

*Given:*

Graph $G(V, E)$ where each vertex $v \in V$ has a type, and each edge $e$ has a weight $w(e)$.

*Output:*

A division of the set $V$ into $k$ subsets $V_1, V_2, V_3, \ldots V_k$ such that:

- Objective function is minimized

- Subject to certain constraints

*Objective:*

Minimize droplet crossover count.

*Constraints:*

- Cutset of the partition is denoted by $T$ and is equal to the set of edges cut by the partition.

- All partitions $V_1 \ldots V_k$ must be of the same size $V/k = n$, and each $V_i, 1 \leq i \leq k$ must maintain the node count for each type.

*Cost function:*

$$Cost = \sum_{e \in T} w(e)$$
where \( w(e) \) is the cost of edge/connection \( e \). Let the partitions be numbered \( 1, 2, \ldots k \) and \( p(u) \) be the partition number of node \( u \). Equivalently, one can write the function \( Cost \) as follows:

\[
Cost = \sum_{e=(u,v) \land p(u) \neq p(v)} w(e)
\]

K-way partitioning is achieved by a series of 2-way optimizations between pairs of DMFBs. Pairwise optimality is only a necessary condition for global optimality. There might be a situation where a complex interchange of three or more nodes from three or more subsets is required to find the global optimum, but there is no efficient way of dealing with that \([47]\). Next, we describe 2-way partitioning using the KL Algorithm.

### 4.5.2 Bi-partitioning using the KL Algorithm \([2]\)

The KL algorithm starts with an initial partition \( (A, B) \) such that \( |A| = n = |B| \), and \( A \cap B = \emptyset \). Let \( P^* = (A^*, B^*) \) be the optimum partition, and \( P = (A, B) \) be the current partition. In order to attain \( P^* \) from \( P \), one has to swap a subset \( X \subseteq A \) with a subset \( Y \subseteq B \) (see Fig. 4.8) such that,

- \( |X| = |Y| \)
- \( X = A \cap B^* \)
- \( Y = A^* \cap B \)
- \( A^* = (A - X) + Y \) and \( B^* = (B - Y) + X \)

The problem of identifying \( X \) and \( Y \) is as hard as that of finding \( P^* = (A^*, B^*) \).

### 4.5.3 Definitions

**Def. 1:** Consider any node \( a \) in block \( A \). The contribution of node \( a \) to the cutset is called the *external cost of \( a \)* and is denoted as \( E_a \), where

\[
E_a = \sum_{v \in B} c_{av}
\]
Def. 2: The internal cost $I_a$ of node $a \in A$ is defined as follows

$$I_a = \sum_{v \in A} c_{av}$$

Def. 3: Moving node $a$ from block $A$ to block $B$ would increase the value of the cutset by $I_a$ and decrease it by $E_a$. Therefore, the benefit of moving $a$ from $A$ to $B$ is

$$D_a = E_a - I_a$$

To maintain a balanced partition, every time we move a node from $A$ to $B$, we must also move a node from $B$ to $A$. The effect of swapping two modules, $a \in A$ and $b \in B$, is characterized by the following lemmas.

Lemma 1: If two elements $a \in A$ and $b \in B$ are interchanged, the reduction in cost (gain) is given by

$$g_{ab} = D_a + D_b - 2c_{ab}$$

Lemma 2: If two elements $a \in A$ and $b \in B$ are interchanged, then the new D-values are given by

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\}$$
CHAPTER 4. SYNTHESIS & OPTIMIZATION

4.5. KERNIGHAN-LIN ALGORITHM

\[ D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\} \]

New values need to be calculated for all the nodes \( x \in A - \{a\} \). Since \( b \) has entered \( A \), the internal cost of \( x \) increases by \( c_{xb} \). Similarly, since \( a \) has entered the opposite block \( B \), the internal cost of \( x \) must be decreased by \( c_{xa} \).

The new internal cost of \( x \) is:

\[ I'_x = I_x - c_{xa} + c_{xb} \]

Similarly, the new external cost of \( x \) is

\[ E'_x = E_x + c_{xa} - c_{xb} \]

Thus, the new D-value of \( x \in A - \{a\} \) is

\[ D'_x = E'_x - I'_x = D_x + 2c_{xa} - 2c_{xb} \]

Similarly, the new D-value for \( y \in B - \{b\} \) is

\[ D'_y = E'_y - I'_y = D_y + 2c_{yb} - 2c_{ya} \]

Algorithm 1 describes bi-partitioning using the KL method. The input subgraph \( G_k \) is the subset of nodes \( (V_k) \) belonging to the two DMFBs being considered, along with all the edges that connect them \( (E_k) \). Note the small changes made to the original algorithm as described in [2] such as choosing \( a \) and \( b \) to be of the same type. This might reduce the overall quality of the cutset but is nonetheless required to keep the architecture of the DMFB intact.

4.5.4 Complexity analysis

Complexity analysis helps us make our algorithm efficient in time (and space). Let us look at Algorithm 1 for this analysis.
Algorithm 1 Modified KL algorithm for bi-partitioning of graphs [2]

**Input:** Undirected subgraph $G_k(V_k, E_k) \subseteq G(V, E)$, $|V_k| = 2n \text{ subgraph has nodes and edges of 2 DMFBs};$

**Output:** Balanced bi-partition $A$ and $B$ with ’small’ cut cost;

1: function $KL\text{-Algo}(G_k(V_k, E_k))$
2: Bipartition $G_k$ into $A$ and $B$ such that $|A| = |B|, A \cap B = \emptyset$ and $A \cup B = V_k$  
   //Each initial subset (A and B) has nodes from the same DMFB
3: repeat
4:   Compute $D_v, \forall v \in V_k$
5:   $A' = A; B' = B, queue \leftarrow \emptyset$
6:   for $i = 0$ to $n$
7:     Find a pair of unlocked vertices $a_i \in A$ and $b_i \in B$ of the same type
     and which maximizes $g_i = D_{a_i} + D_{b_i} - 2c_{a_i b_i}$
8:     Add the pair $(a_i, b_i)$ and gain $g_i$ to queue;
9:     Mark $a_i$ and $b_i$ as locked, $A' = A' - a_i; B' = B' - b_i$ and compute the
     New $D_v$ for all unlocked $v \in V_k$
10:   Find $p$ such that $G_p = \sum_{i=1}^{p} g_i$ is maximized;
11:   if $G_p > 0$ then
12:      Move $X = \{a_1, a_2, ...a_p\}$ from A to B and $Y = \{b_1, b_2, ...b_p\}$ from B to A;
13:     Unlock $v, \forall v \in V$;
14:   until $G_p \leq 0$;

- We require $O(n^2)$ time to calculate/update D values.
- Selecting the best pair $(a_i, b_i)$ from $A$ and $B$ is the costliest exercise which takes $O(n^3)$ time.
   In order to make this step efficient, we arrange D-values in decreasing order.
- Sorting of D values takes $O(n \log n)$ time.
- We begin examining $D_{a_i}$ and $D_{b_j}$ pairwise. If we come across a pair $(D_{a_k}, D_{b_l})$ such that
  $(D_{a_k} + D_{b_l})$ is less than the gain seen so far in our search, we can stop looking further. We
  hardly look at all D values and hence the overall complexity of selecting $(a_i, b_i)$ is limited by
  the sorting process i.e., $O(n \ast \log n)$
- We find $n$ pairs and add them to the queue. Therefore, the complexity of steps 7, 8, 9 in
  algorithm[1] is $O(n^2 \ast \log n)$.
- Step 10 takes linear time to run.
• The number of iterations \( (p) \) makes it \( O(p \times n^2 \times \log n) \) but \( p \) is usually a small number \(< 10\).

For K-way algorithm, we need to repeat the 2-way process \( \binom{k}{2} \) times. Therefore, the overall complexity can be denoted by \( O(k^2 \times p \times n^2 \times \log n) \).

4.5.5 Post processing steps

In a placed DAG, we can see that a set of operations are bound to the same work module/ IO port. When we move the module to a different partition, we need to update each of the operations in that set to the new location. In the above PCR example, operations MIX1, MIX5 and MIX7 were bound to work module 11 (in Fig. 4.6). In case module 11 was swapped during partition, we would have to update all the corresponding operations with the new location.

4.5.6 Case study

We shall briefly give an example to explain the concept clearly. We consider a simple single-split protein assay as shown in Fig. 4.9 that we shall run on a two-DMFB architecture. Each DMFB is of size \( 15 \times 7 \); has two work modules, five input ports and one output port. Let us denote each work module/IO port (hereby known as ‘node’) with a \( W \) prefix. We list each node and all the operations that are bound to them. In our example, we assume \( W_0 \) – \( W_3 \) are detect-type work modules and \( W_4 \) – \( W_{13} \) are IO devices.

In Table 4.1, we can see that the operation DIS1 is bound to node \( W_4 \) (see row 5) and both output operations: \( OUT1 \) and \( OUT2 \) are bound to the same physical entity \( W_8 \) (see line 9). We use Table 4.1 to determine droplet transfer frequency. From Fig. 4.9 and Table 4.1, we can find every instance of a droplet moving from one node to another. For example, DIS1 takes place at node \( W_4 \) and we can see from Fig. 4.9 that the successive operation of DIS1 is MIX1. MIX1 is bound to node \( W_0 \). Therefore, we record a droplet transfer count of 1 for \( W_4 \) and \( W_0 \). In order to represent this information, we create a connectivity matrix as shown in Table 4.2. Any further droplet exchanges between \( W_0 \) and \( W_4 \) updates the count value. The first line shows all the nodes that have droplets transfers with node \( W_0 \). i.e. nodes \( W_1, W_4, W_5, W_8 \) and \( W_{10} \).
Note:

- For faster access, we need to store duplicate information in the connectivity matrix. If there are seven droplets exchanged between $W_0$ and $W_1$ throughout the assay, we store $W_1:7$ in $W_0$'s row and reciprocal information i.e. $W_0:7$ in $W_1$'s row. In this way, we can access their connectivity information faster.

- We only consider those nodes that belong to the chosen pair of DMFBs.

In Table 4.1, we observe that $W_2$ and $W_3$ do not have any operations bound to it. This is a result of the left-edge binding process and is dependent on the nature of the assay.

We start with the following configuration where each part has all the nodes from the same
CHAPTER 4. SYNTHESIS & OPTIMIZATION  

4.5. KERNIGHAN-LIN ALGORITHM

Table 4.1: Post placement Node:Operation binding

<table>
<thead>
<tr>
<th>Node</th>
<th>Associated operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0:</td>
<td>MIX 1, SLT 1, MIX 7, MIX 8, MIX 9, MIX 10, MIX 11, DET2</td>
</tr>
<tr>
<td>W1:</td>
<td>MIX 2, MIX 3, MIX 4, MIX 5, MIX 6, DET 1</td>
</tr>
<tr>
<td>W2:</td>
<td>DIS 1</td>
</tr>
<tr>
<td>W3:</td>
<td>DIS 2, DIS 8, DIS 9, DIS 10, DIS 11</td>
</tr>
<tr>
<td>W4:</td>
<td>DIS 7</td>
</tr>
<tr>
<td>W5:</td>
<td>OUT 1, OUT 2</td>
</tr>
<tr>
<td>W6:</td>
<td>DIS 3, DIS 4, DIS 5, DIS 6</td>
</tr>
<tr>
<td>W7:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W8:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W9:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W10:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W11:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W12:</td>
<td>DIS 12</td>
</tr>
<tr>
<td>W13:</td>
<td>DIS 12</td>
</tr>
</tbody>
</table>

DMFB (bold indicates work modules i.e. non-IO nodes):

\[ PartA : \]

\[ W0, W1, W4, W5, W6, W7, W8 \]

\[ PartB : \]

\[ W2, W3, W9, W10, W11, W12, W13 \]

We calculate internal and external connections (as defined in 4.5.3 Def 1 and Def 2). To find the internal and external connectivity, consider the following example. For node \( W1 \), \( W0 \) is internal as both belong to \( Part A \) and \( W9 \) is external as it is from \( Part B \). From Table 4.1 we see that \( W1 \) has 3 internal (\( W0, W6 \) and \( W8 \)) and 1 external connection (\( W9 \)). \( W1 \)’s internal connectivity sums up to 9 and external connectivity sums up to 4. Sum of all external connections gives us an initial
Table 4.2: Connectivity matrix

<table>
<thead>
<tr>
<th>Node</th>
<th>Connected node and weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0:</td>
<td>W1:7, W4:1, W5:5, W8:1, W10:1</td>
</tr>
<tr>
<td>W1:</td>
<td>W0:7, W6:1, W8:1, W9:4</td>
</tr>
<tr>
<td>W2:</td>
<td></td>
</tr>
<tr>
<td>W3:</td>
<td></td>
</tr>
<tr>
<td>W4:</td>
<td>W0:1</td>
</tr>
<tr>
<td>W5:</td>
<td>W0:5</td>
</tr>
<tr>
<td>W6:</td>
<td>W1:1</td>
</tr>
<tr>
<td>W7:</td>
<td></td>
</tr>
<tr>
<td>W8:</td>
<td>W0:1, W1:1</td>
</tr>
<tr>
<td>W9:</td>
<td>W1:4</td>
</tr>
<tr>
<td>W10:</td>
<td>W0:1</td>
</tr>
<tr>
<td>W11:</td>
<td></td>
</tr>
<tr>
<td>W12:</td>
<td></td>
</tr>
<tr>
<td>W13:</td>
<td></td>
</tr>
</tbody>
</table>

cutset of 5.

**InternalConnections :**

W0 : 14, W1 : 9, W2 : 0, W3 : 0, W4 : 1, W5 : 5, W6 : 1,
W7 : 0, W8 : 2, W9 : 0, W10 : 0, W11 : 0, W12 : 0, W13 : 0

**ExternalConnections :**

W0 : 1, W1 : 4, W2 : 0, W3 : 0, W4 : 0, W5 : 0, W6 : 0,
W7 : 0, W8 : 0, W9 : 4, W10 : 1, W11 : 0, W12 : 0, W13 : 0

These values are then used to find $D$ values as described in sub-section 4.5.3, Def 3. Continuing
our example, we have $W1$’s $D$ value equal to $E_1 - I_1 = 4 - 9 = -5$. We store them as arrays, $D_a$
(for part A) and $D_b$ (for part B), and we use them to determine candidates for exchange based on
equations in [4.5.3] Lemma 1. Nodes are stored in descending order of their $D$ values. The $D$ value
is nothing but the \textit{gain} that we get if we move the node across.

\[ D_a : \]
\[ W7 : 0, W4 : -1, W6 : -1, W8 : -2, W1 : -5, W5 : -5, W0 : -13 \]

\[ D_b : \]
\[ W9 : 4, W10 : 1, W2 : 0, W3 : 0, W11 : 0, W12 : 0, W13 : 0 \]

The best pair from the above \( D \) arrays turns out to be \((W7, W9)\) with a gain of 4. We can observe that any other pair chosen will give a lower gain. We then remove them from future choices and add them to a \textit{Queue} as shown below. After we choose the best pair such that we have the maximum improvement in cut-set, we recalculate \( D \) values based on Lemma 2 in sub-section 4.5.3. The length of \( D \) arrays hence keep decreasing and the \textit{Queue} grows as we find the next best pair and so on. We then find a sub-queue (subset of pairs) to exchange such that the cut-set reduction is maximized. To do this we maintain a cumulative \textit{sum of gains} and the \textit{best gain} so far.

\[ \text{Queue} : \]
\[ W7, W9 : 4 \]
\[ W4, W10 : 0 \]
\[ W6, W11 : -1 \]
\[ W8, W13 : -2 \]
\[ W5, 12 : -5 \]
\[ W1, W2 : -11 \]
\[ W0, W3 : -11 \]
We get the best improvement in cutset if we exchange $W_7$ with $W_9$. Therefore, the cut-set is reduced from 5 to 1. Next, run the same algorithm until there is no further improvement in the cut-set. In this example, the algorithm exits after the second iteration where we do not find any improvement in the cut-set size. Finally, we have the following nodes in each DMFB. Note that the pairs picked for exchange are of the same type as discussed earlier (bold indicates the exchanged nodes).

**Part A:**

$W_0, W_1, W_4, W_5, W_6, W_8, W_9$

**Part B:**

$W_2, W_3, W_7, W_{10}, W_{11}, W_{12}, W_{13}$
Chapter 5

Experiments and Results

This chapter discusses the experimentation setup and pre-requisites along with the results and inferences. We implement the algorithms in C++ and have extensively used the *Digital Microfluidic Biochip Synthesis Framework* [27] from University of California, Riverside, as the base source code available on their website [28] for academic use. We use the Eclipse IDE [48] with MinGW (GNU compiler for windows) [49] for code development. All the experiments were carried out on a laptop with Intel Core i3, 2.1 GHz processor and 4GB RAM.

5.1 Benchmarks

We have three benchmarks available namely, PCR, in-vitro and the protein assay. PCR is a convergent assay as shown in Fig.5.1 but is too small to observe the required behavior on multi-DMFB architectures. In-vitro (Fig. 5.2) is a highly parallel assay and we do not see much of a cut set improvement due to the nature of the assay. Protein assays are divergent assays; they are versatile in the sense that, by increasing the number of split levels, we can create assays of different sizes which allow us to analyze the algorithm better. We shall hence use protein assays for experimentation. *ProteinSplit_x* denotes the set of protein assays with x levels of splits as shown in Fig. 5.3. It results in $2^x$ parallel chains of operations.
CHAPTER 5. EXPERIMENTS AND RESULTS

5.1. BENCHMARKS

Figure 5.1: PCR DAG

Figure 5.2: In-vitro DAG

Figure 5.3: Generic protein DAG [50]

[50]
5.2 Graphs and Notation

The figures in the following sections show time in seconds on the Y-axis. Note that this represents the variable routing time between operations as described in Fig. 4.3. The run-time is based on the assumption that the droplet movement frequency is $100\text{Hz}$ which means that, a droplet can move at a rate of 100 positions per second. $SPPR$ stands for synthesis with partitioning optimization (Scheduling-Placement-Partitioning-Routing) and $SPR$ for synthesis without optimization (Scheduling-Placement-Routing). The notation $2_{15,13}$ denotes a two-DMFB architecture with each DMFB of size $15 \times 13$, and $C = 10$ denotes connector length of 10 electrodes.

5.3 Experiments

The following experiments provide us with a good understanding of the positive and negative effects of partitioning optimization as a phase in the synthesis process. They help us quantify the degree of improvement and figure out various factors that affect run-time and synthesis execution time.

5.3.1 Experiment 1

*Synthesis Runtime Analysis*

Partitioning optimization helps reduce routing time. These are negligible values once assay sizes and architecture sizes increase. However, it is an additional step in the synthesis process and takes up time to execute; apart from the synthesis stages including scheduling, placement and routing. We observe that, unlike scheduling and routing, the partitioning optimization time does not increase rapidly with problem size. For example, consider the data in Table 5.1.

Observe the relatively low partitioning optimization time as compared to the other stages. This shows that we have not added too much overhead to the total synthesis execution time.
Table 5.1: Synthesis execution time for 2_15_13 architecture, C = 30

<table>
<thead>
<tr>
<th>Synthesis stage</th>
<th>ProteinSplit_1 (28 nodes) in ms</th>
<th>ProteinSplit_3 (118 nodes) in ms</th>
<th>ProteinSplit_6 (958 nodes) in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>4</td>
<td>21</td>
<td>164</td>
</tr>
<tr>
<td>Placement</td>
<td>0</td>
<td>8</td>
<td>600</td>
</tr>
<tr>
<td>Partitioning optimization</td>
<td>0</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>Routing</td>
<td>9</td>
<td>73</td>
<td>1914</td>
</tr>
<tr>
<td>Total</td>
<td>13</td>
<td>103</td>
<td>2697</td>
</tr>
</tbody>
</table>

5.3.2 Experiment 2

Routing time vs. Scheduled time We also observe that, as assay size increases, the percentage of routing time increases. This means that routing is an important phase to improve especially if we are thinking of large assays. In Table 5.2, we observe that running proteinSplit_6 we would spend 14% of the total time in routing as compared to a mere 2.7% when we run ProteinSplit_1. Any improvement to the routing time is hence useful, as we would often run the assays multiple times.

Table 5.2: Schedule time vs routing time for 2_15_13 architecture, C = 30

<table>
<thead>
<tr>
<th>Phase</th>
<th>ProteinSplit_1 (28 nodes)</th>
<th>ProteinSplit_3 (118 nodes)</th>
<th>ProteinSplit_6 (958 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>list schedule time (sec)</td>
<td>72</td>
<td>177</td>
<td>1157</td>
</tr>
<tr>
<td>routing time (sec)</td>
<td>2.05</td>
<td>14.02</td>
<td>188.96</td>
</tr>
<tr>
<td>total time (sec)</td>
<td>74.05</td>
<td>191.02</td>
<td>1345.96</td>
</tr>
<tr>
<td>% of routing time</td>
<td>2.76</td>
<td>7.34</td>
<td>14.04</td>
</tr>
</tbody>
</table>

5.3.3 Experiment 3

Compare multi-DMFB architectures to an equivalent-size single DMFB architecture and understand the overhead involved

In this experiment, we keep the parameters such as connector length and the assay type constant and only vary the number of DMFBs. However, the total available work modules \((N_m)\) shall remain the same. Let us maintain \(N_m = 8\) for all three setups as follows, and we shall compare the run-time.

1. One DMFB of size 15x25 with eight modules.
2. Two 15x13 size DMFBs which have four work modules each.

3. Four 15x7 size DMFBs with two work modules each.

We maintain a constant connector length of 30 electrodes. We ran the protein assay with the levels of splits ranging from 1 to 6. The smaller assays i.e. ProteinSplit_(1-3) do not show the overhead effectively since the assays are too small to be split over multiple DMFBs. The algorithm inherently uses as few DMFBs as possible. This allows us to decide on how big a setup we would need for the assay. In Fig. 5.4, we can see a smaller increase in routing time, going from single to double DMFB then to four-DMFB architecture. This shows that adding more DMFBs than necessary can be deterrent to routing time. We would ideally use as few and as big DMFBs as possible; especially when we are using longer connectors. The disparity generally increases as the size of the assay increases and it is worse if we do not do partitioning optimization, as seen in Fig. 5.5.
5.3.4 Experiment 4

*Compare the maximum size of the assay that can run on different DMFB architectures (varying number of DMFBs)*

Let us discuss a scenario where we have only DMFBs of size 15x7 but we have many of them. We would like to find the largest assay that we can run using the apparatus at hand. For this experiment, we ran ProteinSplit_(1-6) and we found that as we increased the number of DMFBs in the architecture, we could run bigger assays (see Fig. 5.6). With a two-DMFB setup, we could only run up to 3-level split protein assay whereas, in a three-DMFB setup, we ran up to 5-level split protein assay and in a four-DMFB setup, we could run all 6 protein assays and probably more. *The ability to run larger assays is one of the fundamental reasons for opting for a multi-DMFB approach.* Note that we have kept the connector length constant at 30. The three and four-DMFB architectures that we used are as shown in Fig. 5.5(b) and (c) respectively. Changing the architecture might change the performance.
5.3.5 Experiment 5

*Observe the effect of changing the connector length and compare the difference with and without partitioning optimization.*

In practice, we would need to place DMFBs close by and connect them as discussed in chapter 3 but, we might need to use longer connectors to accommodate more DMFBs. Here, we discuss a simple experiment to see the effect of increasing connector length on total routing time. We sweep the connector length values from 10 to 100 in steps of 10 and observe the effect on multiple protein assays. Fig. 5.7 shows the linear increase in routing time for protein assays on the 2_15_7 architecture. We see that the *slope* increases as the complexity of the assay increases. It implies that, for a smaller assay, change in connector length has lesser effect. Fig. 5.8 shows a similar figure for the 4_15_7 architecture. We can see the same trend with larger assays as well. The important takeaway from this exercise is that we understand the effect of connector length on

![Figure 5.6: Number of DMFBs](image-url)
5.3.6 Experiment 6

Observe the effect of varying the number of copies of the assay being run simultaneously on certain architecture

It is common to find multiple copies of the same assay being run on the DMFB. In this experiment, we increased the number of copies of the ProteinSplit_3 assay from 1 to 10 and observed the behavior, with and without partitioning optimization. We used the 2_15_13 architecture setup for this experiment with a connector length of 30. In Fig. 5.9, we can observe that, with partitioning optimization, the routing time increases slower than without optimization. This shows that partitioning optimization becomes more important as problem size increases. The slope is smaller for SPPR. We maintain the connector length to be constant at 30. We notice that increasing number of copies of the same DAG causes linear increase in routing time whereas, increasing the number of split levels increases the routing time exponentially, as observed in Fig. 5.4.
CHAPTER 5. EXPERIMENTS AND RESULTS

5.3. EXPERIMENTS

Figure 5.8: Connector length sweep for 4_15_7 architecture

Figure 5.9: Multiples of ProteinSplit_3 DAG
5.4 Result interpretation

We can make the following conclusions from the above experiments

1. The larger the problem size is, the larger the required number of work modules will be.

2. Increasing connector length affects run time linearly.

3. Increasing connector length has a higher impact for larger assays.

4. Increasing the number of DAGs running simultaneously on the DMFB architecture shows linear increase in routing time.

5. The effect of increasing the number of DAGs running simultaneously on the DMFB architecture can be made better through partitioning optimization.

6. By adding more DMFBs, we can run larger assays

7. Multi-DMFB architecture causes more overhead in run time as compared to single DMFB with the same number of work modules ($N_m$ being equal).
Chapter 6

Conclusions and Suggestions for Future Research

The multi-DMFB architecture and associated synthesis methodology, as discussed in this thesis, have added a new dimension to scalable DMFB designs. The availability of the required hardware, such as a connector, adds to the validity of the idea of using multiple DMFBs as an alternative to increasing its size. As an added improvement, we have suggested a droplet crossover minimization technique based on the well-known KL algorithm (used in VLSI circuit design) that further optimizes the runtime. Experimental results show that, larger architectures can handle larger assays with ease. Routing times can be improved by droplet exchange minimization. As the problem size increases, we found that the ratio of routing time to scheduled assay time increases. We varied different parameters such as connector length, number of DMFBs, number of split levels in the protein assay and number of DAGs running at once and reported the results. Experimental results demonstrate the feasibility of the proposed multi-DMFB approach, before very large-scaled DMFB can be realized.

Future work could focus on making the existing multi-DMFB synthesis generic in terms of type and size of DMFBs. Different scheduling algorithms can be modified and results can be compared for multi-DMFB architectures. Lastly, new partitioning optimization schemes can be found to better reduce the droplet crossover count.
Bibliography


