System-Level Modeling and Synthesis of Flow-Based Microfluidic Biochips

Wajid Hassan Minhass
whmi@imm.dtu.dk

Paul Pop
pop@imm.dtu.dk

Jan Madsen
jan@imm.dtu.dk

DTU Informatics
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark

ABSTRACT

Microfluidic biochips are replacing the conventional biochemical analyzers and are able to integrate the necessary functions for biochemical analysis on-chip. There are several types of microfluidic biochips, each having its advantages and limitations. In this paper we are interested in flow-based biochips, in which the flow of liquid is manipulated using integrated microvalves. By combining several microvalves, more complex units, such as micropumps, switches, mixers, and multiplexers, can be built. Although researchers have proposed significant work on the system-level synthesis of droplet-based biochips, which manipulate droplets on a two-dimensional array of electrodes, no research on system-level synthesis of flow-based biochips has been reported so far. The focus has been on application modeling and component-level simulation. Therefore, for the first time to our knowledge, we propose a system-level modeling and synthesis approach for flow-based biochips. We have developed a topology graph-based model of the biochip architecture, and we have used a sequencing graph to model the biochemical applications. We consider that the architecture of the biochip is given, and we are interested to synthesize an implementation, consisting of the binding of operations in the application to the functional units of the architecture, the scheduling of operations and the routing and scheduling of the fluid flows, such that the application completion time is minimized. We propose a List Scheduling-based heuristic for solving this problem. The proposed heuristic has been evaluated using two real-life case studies and a set of four synthetic benchmarks.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design, Performance

Keywords

Microfluidics, biochips, modeling, synthesis

CASES'11, October 9–14, 2011, Taipei, Taiwan.

1. INTRODUCTION

Microfluidics-based biochips have become an actively researched area in recent years. Sometimes also referred to as lab-on-a-chip, biochips integrate different biochemical analysis functionalities (e.g., dispensers, filters, mixers, separators, detectors) on-chip, miniaturizing the macroscopic chemical and biological processes to a submillimetre scale [17]. These microsystems offer several advantages over the conventional biochemical analyzers, e.g., reduced sample and reagent volumes, speeded up biochemical reactions, ultra-sensitive detection and higher system throughput, with several assays being integrated on the same chip [20].

Microfluidic biochips can readily facilitate clinical diagnostics, especially immediate point-of-care disease diagnosis. In addition, they also offer exciting application opportunities in the realm of massively parallel DNA analysis, enzymatic and proteomic analysis, cancer and stem cell research, and automated drug discovery [17, 5]. Utilizing these biochips to perform food control testing, environmental (e.g., air and water samples) monitoring and biological weapons detection are also interesting possibilities.

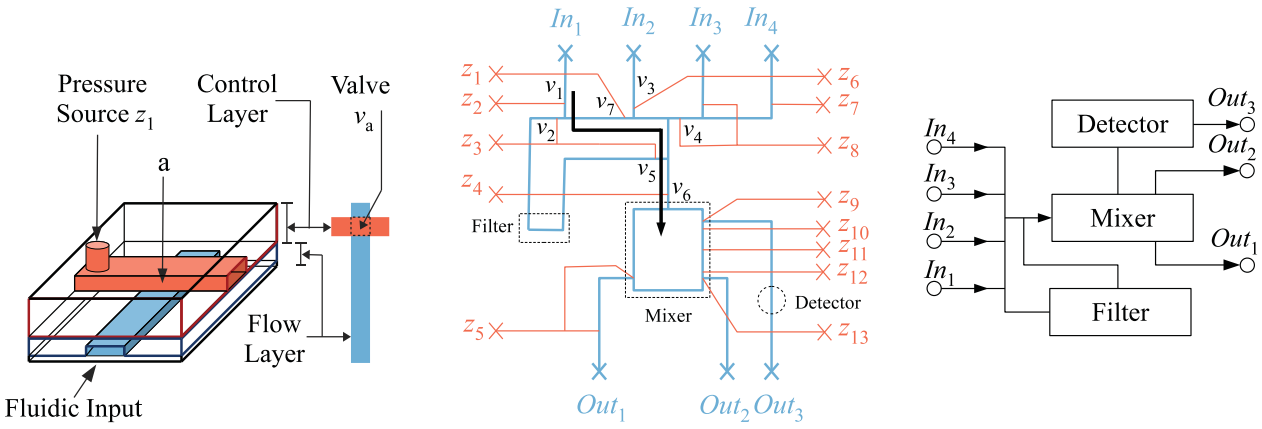
There are several types of microfluidic biochip platforms, each having its own advantages and limitations [10]. In this paper, we focus on the flow-based biochips in which the microfluidic channel circuitry on the chip is equipped with chip-integrated microvalves that are used to manipulate the on-chip fluid flow [17]. By combining several microvalves, more complex units like mixers, micropumps, multiplexers etc. can be built up, with hundreds of units being accommodated on one single chip [10].

1.1 Related Work

During the last decade, a significant amount of work has been carried out on the individual microfluidic components as well as the microfluidic platforms [9, 10]. The manufacturing technology, soft lithography, used for the flow-based biochips has advanced faster than Moore's law [6]. Although biochips are becoming more complex everyday, Computer-Aided Design (CAD) tools for these chips are still in their infancy. Most CAD research has been focussed on device-level physical modeling of components [13, 7]. Designers are using full-custom and bottom-up methodologies involving many manual steps to implement these chips.

Currently, researchers manually map the applications to the valves of the chip using some custom interface (analogous to exposure of gate-level details) [16]. The manual process is quite tedious and needs to be repeated every time a change is made either to the chip architecture or the biochemical application. For larger chips and applications, the process can easily result in inefficient architectural mappings.

As the chips grow more complex (commercial biochips are available which use more than 25,000 valves and about a million fea-



(a) Microfluidic Valve (b) Biochip: Schematic View (c) Biochip: Functional View

Figure 1: Flow-Based Biochip Architecture

tures to run 9,216 polymerase chain reactions in parallel [12]) and the need of having multiple and concurrent assays on the chip becomes more significant, these methodologies become highly inadequate [3]. Therefore, new top-down design methodologies and design tools are needed, in order to provide the same level of CAD support to the biochip designer as the one currently taken for granted in the semiconductor industry [3].

Researchers have proposed significant work on top-down synthesis methodologies for droplet-based biochips [2]. In these chips, the liquid is manipulated as discrete droplets on an electrode array. The synthesis process, starting from a given biochemical application and a droplet-based biochip architecture model, determines the resource allocation, binding, scheduling and placement of the application operations. However, these techniques are not applicable to the flow-based chips and, to the best of our knowledge, no system-level modeling and synthesis approach has been proposed so far for the flow-based biochips.

1.2 Contribution

We propose a topology graph-based system-level model of a biochip architecture, that is independent of the underlying biochip implementation technology. Using the proposed model, we focus on the problem of synthesizing a biochemical application, modeled as a sequencing graph (capturing the operations and their dependency constraints), onto a given biochip architecture. We propose a List Scheduling-based binding and scheduling heuristic aiming at reducing the application completion time. In microfluidic biochips, routing latencies are comparable to the operation execution times, thus having a considerable influence on the schedule. Our heuristic also takes the routing and fluid contention into account. As an output of the synthesis process, we generate the control sequence for a biochip controller in order to execute the biochemical application onto the specified biochip. We evaluate the proposed framework by synthesizing two real-life case studies as well as a set of four synthetic benchmarks.

The paper is organized in six sections. We present the proposed component model and the biochip architecture model in Section 2.1. Section 2.2 covers the abstract biochemical application model. The targeted problem is discussed and formulated in Section 3. The proposed synthesis approach is presented in Section 4 and is further evaluated in Section 5. We close by presenting our conclusions (Section 6).

2. SYSTEM MODEL

2.1 Biochip Architecture Model

Fig. 1b shows the schematic view of a flow-based biochip with four input ports and three output ports, a mixer, a filter and a detector. Fig. 1c shows the functional level view of the same chip. The biochip is manufactured using multilayer soft lithography [17]. A cheap, rubber-like elastomer (polydimethylsiloxane, PDMS) with good biocompatibility and optical transparency is used as the fabrication substrate. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: *flow layer* (depicted in blue) and the *control layer* (depicted in red) [17]. The liquid in the flow layer is manipulated using the control layer.

The basic building block of such a biochip is a valve (see Fig. 1a), which is used to manipulate the fluid in the flow layer as the valves restrict/ permit the fluid flow. The control layer (red) is connected to an external air pressure source z_1. The flow layer (blue) is connected to a fluid reservoir through a pump that generates the fluid flow. When the pressure source is not active, the fluid is permitted to flow freely (open valve). When the pressure source is activated, high pressure causes the elastic control layer to pinch the underlying flow layer (point a in Fig. 1a) blocking the fluid flow at point a (closed valve). Because of their small size ($100 \times 100\ \mu m^2$), a biochip can accommodate thousands of valves. Analogous to its microelectronics counterpart, this approach is called microfluidic Large Scale Integration (LSI) [17].

By combining these valves, more complex units, such as switches, multiplexers, micropumps, mixers, can be built [10]. For example,

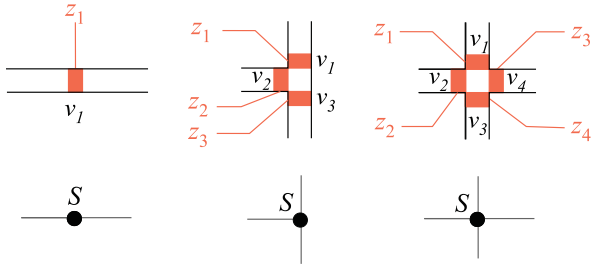


Figure 2: Switch Configurations

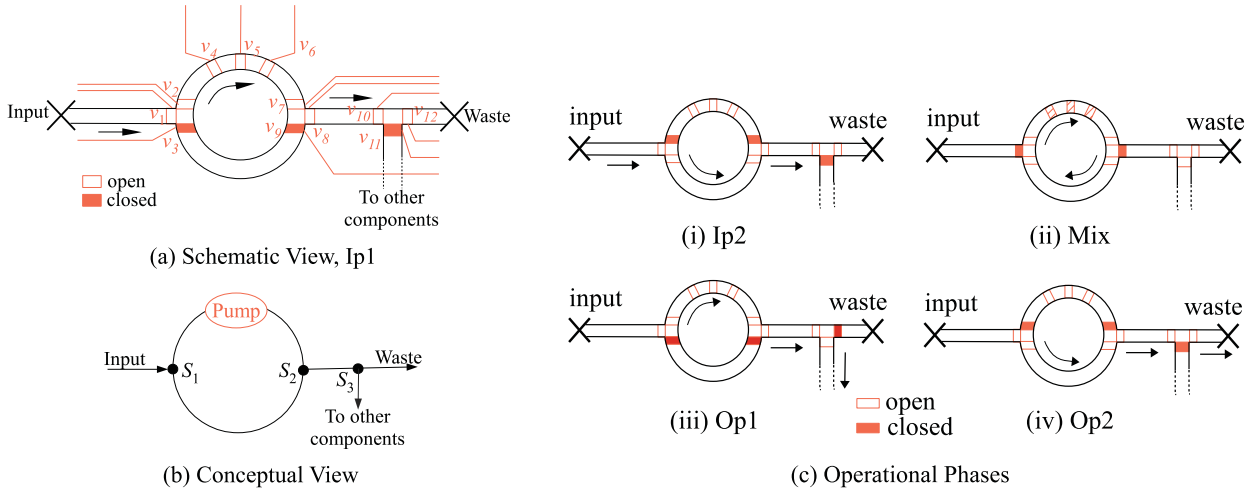


Figure 3: Microfluidic Mixer

the valves can be combined to represent a switch. As shown in Fig. 2, a switch may consist of one valve (restricting/ allowing flow in a channel) or may consist of more than one valve. Multiple valve switches are present at the channel junctions and are used to control the path of the fluids entering the switch from different sides. The fluid flow can be generated using off-chip or on-chip pumps. The path of the fluid flow is established using the microfluidic valves. For example, in Fig. 1b, in order to create a flow path from In_1 to the Mixer unit, valves v_2, v_3, v_4, v_5 need to be closed, while valves v_1, v_6 and v_7 must be left open. A pumping action at In_1 would then direct the fluid to flow through the established path towards the Mixer.

2.1.1 Component Model

Consider the pneumatic mixer [4] in Fig. 3a which is implemented using nine microfluidic valves, v_1 to v_9. Fig. 3b shows the conceptual view of the same mixer. The valve set $\{v_4, v_5, v_6\}$ acts as an on-chip pump. The valve set $\{v_1, v_2, v_3\}$ is termed as switch S_1 and the valve set $\{v_7, v_8, v_9\}$ as switch S_2. The two switches facilitate the inputs and outputs, and the pump is used to perform the mixing. The mixer output can either be sent to the waste or to the other components in the chip using the switch S_3, as shown by the paths in Fig. 3a.

The mixer has five operational phases. The first two phases represent the input of two fluid samples that need to be mixed, followed by the mixing phase. The mixed sample is then transported out of the mixer in the last two phases. For the first fluidic input (phase Ip1, depicted in Fig. 3a), valves v_1, v_2, v_7 and v_8 are opened (together with v_4, v_5, v_6), the pump at the *Input* is activated and the liquid fills in the upper half of the mixer.

Note that the fluid samples that are to be mixed do not need to occupy the full channel length from the *Input* to the upper half of the mixer. Rather each sample occupies a certain length on the flow channel. The process of measuring the length of each fluid sample is called *metering* and is carried out by transporting the sample between two valves that are a fixed length apart [19].

In general, in order to avoid fluid dispersion, the samples are not directly placed in the channel [19]. Instead, the samples are immersed in a filler fluid (e.g., immiscible mineral oil) in order to be transported from one part of the chip to the other. The fluid sample volumes can be precisely controlled. In order for the filler fluid (oil) to flow on the flow layer, the point of flow origin needs to be connected to an oil reservoir and the point of flow termination needs to be connected to a waste outlet where the oil can be collected after traversing through the chip. Also, an on-chip or off-chip pump source needs to be available in order to generate the flow. All chip input ports are generally connected to a pump and the oil reservoirs. In Fig. 3a, a flow of fluid from the input port to the upper half of the mixer would not have been possible if the mixer output was not connected to the waste outlet as there would be no closed loop for the oil to flow in. But since the mixer has a waste outlet, the oil flows from the input, goes through the mixer and into the waste outlet. The fluid sample that needs to be mixed rides over the oil and reaches the mixer. Once the top half is filled, the valves v_7 and v_2 close, stopping the oil flow and blocking the fluid sample in the upper half of the mixer. Note that, since we know the flow rate (mm/s) and the sample volume (in mm, measured in terms of length through *metering*), the time until the mixer gets filled can be easily calculated, hence an optical feedback is not necessary in order to activate the valves at the right moment.

In the next phase Ip2, the second fluid sample fills the lower half of the mixer (Fig. 3c-i). Once both halves are filled, the mixer input and output valves (v_1 and v_8) are closed while valves v_2, v_3, v_7, v_9 are opened and the mixing operation is initiated (Fig. 3c-ii). Valve set $\{v_4, v_5, v_6\}$ acts as a peristaltic pump. Closing valve v_4 inserts some pressure on the fluid inside the mixer, closing valve v_5 creates further pressure, then as valve v_6 is closed valve v_4 is opened again. This forces the liquid to rotate clockwise in the mixer. The valves are closed and opened in a sequence such that the liquid rotates at a certain speed accomplishing the mixing operation. Next, in phase Op1 (Fig. 3c-iii), half of the mixed sample is pushed out of the mixer towards the rest of the chip and in Op2 (Fig. 3c-iv), the other half is transported to the waste.

Using pressurized microfluidic valves in the control layer is the most commonly utilized control method for the flow-based biochips. However, microfluidic components equipped with alternate control techniques (e.g., electro-osmotic, electrokinetic) have also been developed. In order to have a unified design methodology covering several underlying technologies, it is imperative to model the component implementation technology details separately from its operational capabilities.

We propose a dual-layer component modeling framework, consisting of a *flow layer model* and a *control layer model*. The flow layer model $(\mathcal{P}, C)$ of each component $\mathcal{M}$ is characterized by a set of operational phases $\mathcal{P}$ and execution time C. The control layer model captures the valve actuation details required for the on-chip execution of all operational phases of a component. For example, Table 1 presents

Table 1: Mixer: Control Layer Model

Phase	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
1. Ip1	0	0	1	0	0	0	0	0	1
2. Ip2	0	1	0	0	0	0	1	0	0
3. Mix	1	0	0	Mix	Mix	Mix	0	1	0
4. Op1	0	0	1	0	0	0	0	0	1
5. Op2	0	1	0	0	0	0	1	0	0

Table 2: Component Library ($\mathcal{L}$): Flow Layer Model

Component	Phases (P)	Execution Time (C)
Mixer [9]	Ip1/ Ip2/ **Mix**/ Op1/ Op2	0.5s
Filter [9]	Ip/ **Filter**/ Op1/ Op2	20s
Detector [9]	Ip/ **Detect**/ op	5s
Separator [9]	Ip1/ Ip2/ **Separate**/ Op1/ Op2	140s
Heater [8]	Ip/ **Heat**/ Op	20°C/s

the control layer model of a pneumatic mixer, as presented in Fig. 3, whose flow layer model is characterized by the first row in Table 2. In Table 1, the valve activation for each phase is shown, '0' representing an open and '1' a closed valve. The status 'Mix' shown for the valve set $\{v_4, v_5, v_6\}$ on line 4 of Table 1 represents the mixing step in which these valves are opened and closed in a specific sequence to achieve mixing. Microfluidic platforms are equipped with a controller that manages all on-chip control, i.e., issuing signals to on-chip components for executing a biochemical application, performing data acquisition and signal processing operations [9]. The control layer model of a component contains all the details that a biochip controller requires for executing the operational phases of that component.

Table 2 shows the flow layer model library $\mathcal{L} = \mathcal{M}(\mathcal{P}, C)$ of five commonly utilized microfluidic components. The different operational phases listed for a component may or may not be executable in parallel depending on how the component is implemented, e.g., the mixer presented here has only one input port to receive both the input fluids, thus only one input phase can be activated at a time.

2.1.2 Architecture Model

Most of the research work carried out for modeling the microfluidic architecture has been focused on the device-level physical models [7, 13]. These are inadequate for system-level top-down design methodologies and new modeling frameworks need to be developed. We propose a system-level model based on a topology graph in order to capture the biochip architecture.

Consider the biochip architecture shown in Fig. 4. The chip has two inputs, two outputs and is equipped with three mixers, one heater, one filter and eight storage reservoirs, i.e., the component 'Storage-8' contains eight reservoirs, Res_1–Res_8. The biochip architecture is modeled as a topology graph $\mathcal{A} = (\mathcal{N}, \mathcal{S}, \mathcal{D}, \mathcal{F}, c)$, where $\mathcal{N}$ is a finite set of vertices, $\mathcal{S}$ is a subset of $\mathcal{N}$, $\mathcal{S} \subseteq \mathcal{N}$, $\mathcal{D}$ is a finite set of directed edges and $\mathcal{F}$ is a finite set of flow paths. A vertex $N \in \mathcal{N}$ has two distinguished types: a vertex $S \in \mathcal{S}$ represents a switch (e.g., S_1 in Fig. 4), whereas a vertex $M \in \mathcal{N}, \notin \mathcal{S}$, represents a component or an input/output node (e.g., $Mixer_1$ and In_1, respectively, in Fig. 4). A directed edge $D_{i,j} \in \mathcal{D}$ represents a directed communication channel from the vertex N_i to vertex N_j, with $N_i, N_j \in \mathcal{N}$ (e.g., D_{In_1,S_1} represents a directed link from vertex In_1 to vertex S_1). A flow path, $F \in \mathcal{F}$, is a subset of two or more directed edges of $\mathcal{D}$, $F \subseteq \mathcal{D}$, $|F| > 1$, representing a directed communication link between any two vertices $\in \mathcal{N}$ using a chain of directed edges of $\mathcal{D}$ (e.g., $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$ represents a directed link from vertex In_1 to vertex $Mixer_1$). The weight c associated with a directed edge $D \in \mathcal{D}$ or a flow path $F \in \mathcal{F}$ represents its transport latency.

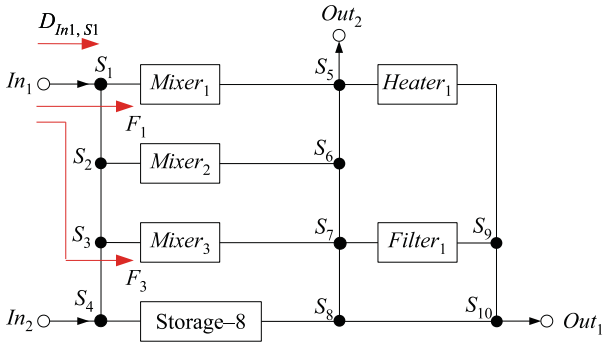


Figure 4: Biochip Architecture

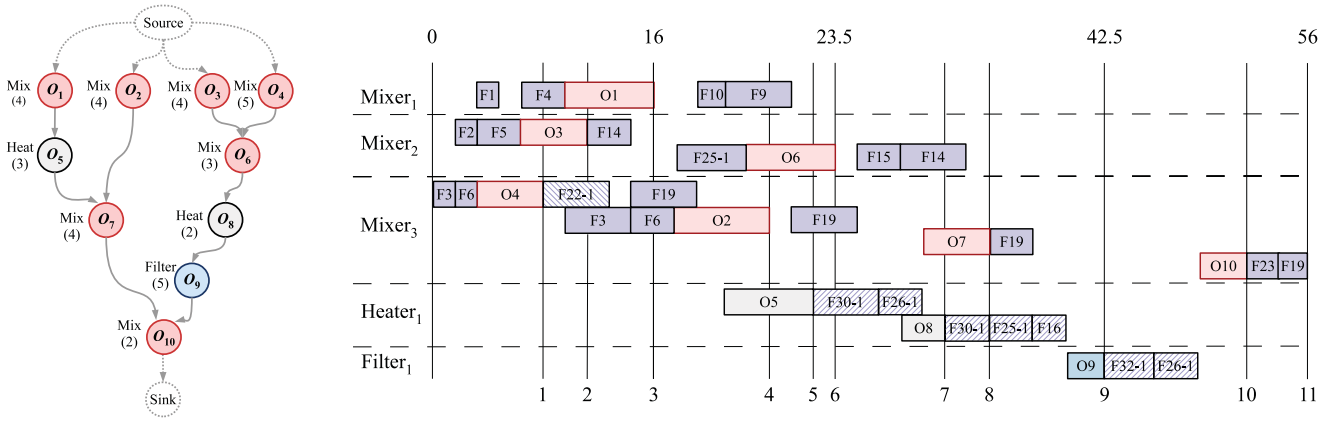
The set of flow paths $\mathcal{F}$ is the set of permissible flow routes on the biochip. These flow paths are specified by the biochip designer but can be easily extracted from the chip architecture as well, if all pump locations, oil inlet and waste outlet locations are known. As discussed in the previous section, in order for a route to be permissible on the chip (e.g., Fig. 4., flow from chip input In_1 to the mixer $Mixer_1$, $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$), the point of flow origin (In_1) needs to be connected to a pump (and an oil reservoir) and the point of flow termination ($Mixer_1$) needs to be connected to a waste output. This means that in order for independent two way flows (one direction at a time since the same channel is used for the flow in both directions) to be possible between any two vertices, e.g., S_1 to S_2, both S_1 and S_2 would need to have an independent connection to a pump, a waste outlet and an oil inlet. Providing such an independent connection to every vertex requires a very high number of pumps, oil inlets and waste outlets. Thus only a limited number of vertices are provided with these connections, limiting the number of allowed flows on the chip. The allowed flows are captured by the set of flow paths $\mathcal{F}$. Each route (flow path) has an associated control layer model that contains the details required for its utilization, i.e., the switch sequence and the pump activation details.

Analogous to a circuit-switched network, the entire flow route is reserved until the completion of the fluid transfer (e.g., until the fluid reaches $Mixer_1$), imposing routing constraints on the chip. These constraints can be extracted from the set $\mathcal{F}$. All those flow paths in the set $\mathcal{F}$ that have a network vertex N_i in common as a source or destination vertex of any of their directed edges D, are considered as mutually exclusive, i.e., the routes represented by these flow paths can only be utilized in a serialized fashion. For example in Fig. 4, $F_{In_1,Mixer_1}$ and $F_{In_1,Mixer_2}$ are mutually exclusive as they share the vertices In_1 and S_1.

Since fluid samples are expendable and cannot be reused limitless number of times (unlike the operands in computers), the fluid volumes need to be managed inside the chip. We assume that the designer does this beforehand while designing the biochemical application [1], ensuring that both overflow and underflow are avoided.

2.2 Biochemical Application Model

We model a biochemical application using a sequencing graph [3]. The graph $\mathcal{G}(O, \mathcal{E})$ is directed, acyclic and polar (i.e., there is a *source* vertex that has no predecessors and a *sink* vertex that has no successors). Each vertex $O_i \in O$ represents an operation that can be bound to a component using a binding function $\mathcal{B} : O \rightarrow \mathcal{M}$. Each vertex has an associated weight $C_i^{M_j}$, which denotes the execution time required for the operation O_i to be completed on component M_j.



(a) Application Graph (b) Schedule

Figure 5: Illustrative Example

The execution times provided in Table 2 are of the actual functional phase (given in bold in the table, e.g., **Mix**). These execution times are taken as the typical execution times for the said component types, i.e., typical mixing time is 0.5 s but a biochemical application description may specify a longer time (e.g., 5 s) if required for a particular operation. This value does not include the time required to fetch the input fluids or to remove the output fluids from the component. The input/output (I/O) phases are dependent on the chip architecture and are thus captured by the set of flow paths $\mathcal{F}$ in the biochip architecture model $\mathcal{A}$. The edge set $\mathcal{E}$ models the dependency constraints in the assay, i.e., an edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of O_i is the input of O_j. All inputs need to arrive before an operation can be activated. We assume that the biochemical application has been correctly designed, such that all operations will have the correct volume of liquid available for their execution. Fig. 5a shows an example of a biochemical application model which has seven mixing operations (O_1–O_4, O_6, O_7, O_{10}), one filtration operation (O_9) and two heating operations (O_5, O_8). The execution times for the operations are given in Fig. 5a (the parameter below the operation type).

3. BIOCHIP SYNTHESIS

Implementing a biochemical application onto a biochip architecture requires the following steps: *allocation* of components from a library $\mathcal{L}$, the *placement* of components on a given area, *binding* of operations onto the allocated components, *scheduling* the operations and performing the required fluidic *routing*. The following subsections explain these design tasks using the biochip architecture given in Fig. 4 and the biochemical application in Fig. 5a.

3.1 Allocation and Placement

In this paper, we consider that the architecture is given, i.e., the allocation and placement steps have already been performed. The architecture is modeled as described in Section 2.1. Thus, the allocated components are captured by the vertex set $\mathcal{M}$, $\mathcal{M} \in \mathcal{N}$, in the architecture model $\mathcal{A}$. Table 3 shows the set $\mathcal{M}$ for the biochip given in Fig. 4. The component placement and interconnections are also given, and are captured by the remaining elements of the topology graph $\mathcal{A}$ modeling the architecture.

Table 3: Allocated Components ($\mathcal{M}$)

Function	Units	Notations
Input port	2	In_1, In_2
Output port	2	Out_1, Out_2
Mixer	3	$Mixer_1$, $Mixer_2$, $Mixer_3$
Heater	1	$Heater_1$
Filter	1	$Filter_1$
Storage Reservoir	8	Res_1–Res_8

3.2 Binding, Scheduling and Routing

Fig. 5b shows the schedule for executing the biochemical application in Fig. 5a on the biochip architecture in Fig. 4. The schedule is represented as a Gantt chart, where, we represent the operations and fluid routing phases as rectangles, with their lengths corresponding to their execution duration. Each operation is placed in a separate row. During the binding step, each vertex O_i, $O_i \in O$, representing a biochemical operation in the application model in Fig. 5a is bound to an available component M_j, i.e., $\mathcal{B}(O_i) = M_j$. For example, the mixing operation O_1 in the application model in Fig. 5a is bound to the component $Mixer_1$ as shown in Fig. 5b. Since the fluid transport latencies in microfluidic chips are comparable to the operation execution times, fluid routing also needs to be considered during the synthesis phase. This means that the binding function must also capture the binding of the edge set $\mathcal{E} \in \mathcal{G}$ to an available route. The available route can be a flow path, $F \in \mathcal{F}$, or a collection of flow paths called a *composite route*. A composite route is used if the source and destination components are such that no direct flow path exists between them.

A scheduling strategy is needed to efficiently execute the biochemical operations on the chip components, while considering the dependency and resource constraints captured by the biochemical application and the biochip architecture models, respectively. In Fig. 5b, operation O_6 bound to $Mixer_2$ starts immediately after all its predecessors (O_3, O_4) are complete and the input fluids have been routed to $Mixer_2$. It starts at $t_{O_6}^{start} = 20.5$ s and takes 3 s, finishing at time $t_{O_6}^{finish} = 23.5$ s.

The different phases during the execution of an operation O_i on a component M_j are captured by the set of operational phases, P, in the component model (Table 2). The time needed to execute an operation O_i on component M_j, $C_i^{M_j}$, is given by the application model $\mathcal{G}$. The input/output (I/O) phases are dependent on the chip architecture and are thus captured by the set of flow paths $\mathcal{F}$ in the biochip architecture model $\mathcal{A}$.

Together with the set of operations $O \in \mathcal{G}$ given in the application model, the edge set $\mathcal{E} \in \mathcal{G}$ also needs to be scheduled on

Table 4: Flow Path Set ($\mathcal{F}$) and Routing Constraints

$F_1 = (In_1, S_1, Mixer_1)$, 2 s	$F_{18} = (Mixer_2, S_6, S_7, S_8, S_{10}, Out_1)$, 3.5 s	*Routing Constraints:*
$F_2 = (In_1, S_1, S_2, Mixer_2)$, 2.5 s	$F_{19} = (Mixer_3, S_7, S_6, S_5, Out_2$, 3 s	
$F_3 = (In_1, S_1, S_2, S_3, Mixer_3)$, 3 s	$F_{20} = (Mixer_3, S_7, S_6, S_5, Heater_1)$, 3 s	$F_1 : F_2 \vee F_3 \vee F_4 \vee F_7 \vee F_{24}$
$F_4 = (In_2, S_4, S_3, S_2, S_1, Mixer_1)$, 3.5 s	$F_{21} = (Mixer_3, S_7, Filter_1)$, 2 s	$F_2 : F_1 \vee F_3 \vee F_4 \vee F_5 \vee F_7 \vee F_{24} \vee F_{25}$
$F_5 = (In_2, S_4, S_3, S_2, Mixer_2)$, 3 s	$F_{22-x} = (Mixer_3, S_7, S_8, Storage$-8), 2.5 s	$F_3 : F_1 \vee F_2 \vee F_4 \vee F_5 \vee F_6 \vee F_7 \vee F_{24}$
$F_6 = (In_2, S_4, S_3, Mixer_3)$, 2.5 s	$F_{23} = (Mixer_3, S_7, S_8, S_{10}, Out_1)$, 3 s	$\vee F_{25} \vee F_{26}$
$F_{7-x} = (In_1, S_1, S_2, S_3, S_4, Storage$-8), 3.5 s	$F_{24-x} = (Storage$-8, $S_4, S_3, S_2, S_1, Mixer_1)$, 3.5 s	$F_4 : F_1 \vee F_2 \vee F_3 \vee F_5 \vee F_6 \vee F_7 \vee F_8$
$F_{8-x} = (In_2, S_4, Storage$-8), 2 s	$F_{25-x} = (Storage$-8, $S_4, S_3, S_2, Mixer_2$, 3 s	$\vee F_{24} \vee F_{25} \vee F_{26}$
$F_9 = (Mixer_1, S_5, Out_2)$, 2 s	$F_{26-x} = (Storage$-8, $S_4, S_3, Mixer_3)$, 2.5 s	$F_5 : F_2 \vee F_3 \vee F_4 \vee F_6 \vee F_7 \vee F_8 \vee F_{24}$
$F_{10} = (Mixer_1, S_5, Heater_1)$, 2 s	$F_{27-x} = (Storage$-8, $S_8, S_7, S_6, S_5, Heater_1)$, 3.5 s	$\vee F_{25} \vee F_{26} \vee F_{27}$
$F_{11} = (Mixer_1, S_5, S_6, S_7, Filter_1)$, 3 s	$F_{28-x} = (Storage$-8, $S_8, S_7, Filter_1)$, 2.5 s	$F_6 : F_3 \vee F_4 \vee F_5 \vee F_7 \vee F_8 \vee F_{24} \vee F_{25}$
$F_{12-x} = (Mixer_1, S_5, S_6, S_7, S_8, Storage$-8), 3.5 s	$F_{29-x} = (Storage$-8, $S_8, S_{10}, Out_1)$, 2.5 s	$\vee F_{26}$
$F_{13} = (Mixer_1, S_5, S_6, S_7, S_8, S_{10}, Out_1)$, 4 s	$F_{30-x} = (Heater_1, S_9, S_{10}, S_8, Storage$-8), 3 s	$F_7 : F_1 \vee F_2 \vee F_3 \vee F_4 \vee F_5 \vee F_6 \vee F_8$
$F_{14} = (Mixer_2, S_6, S_5, Out_2)$, 2.5 s	$F_{31} = (Heater_1, S_9, S_{10}, Out_1)$, 2.5 s	$\vee F_{24} \vee F_{25} \vee F_{26}$
$F_{15} = (Mixer_2, S_6, S_5, Heater_1)$, 2.5 s	$F_{32-x} = (Filter_1, S_9, S_{10}, S_8, Storage$-8), 3 s	...
$F_{16} = (Mixer_2, S_6, S_7, Filter_1)$, 2.5 s	$F_{33} = (Filter_1, S_9, S_{10}, Out_1)$, 2.5 s	$F_{33} : F_{13} \vee F_{18} \vee F_{23} \vee F_{29} \vee F_{30} \vee F_{31}$
$F_{17-x} = (Mixer_2, S_6, S_7, S_8, Storage$-8), 3 s		$\vee F_{32}$

the chip, while taking the routing constraints into account. Table 4 shows the flow path set (permissible route set), $\mathcal{F}$, for the biochip given in Fig. 4. A shorter representation (using the vertices traversed in the flow path) is chosen for clarity, for example, the flow path $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$ is represented as $F_1 = (In_1, S_1, Mixer_1)$. Also, note that each flow path involving the storage reservoir (e.g., F_{7-x}) represents a set of eight flow paths (F_{7-1} to F_{7-8}), i.e., one for each of the eight storage reservoirs. The routing constraints (as discussed in Section 2.1.2) extracted from the set are also shown in Table 4. The first row in the routing constraints ($F_1 : F_2 \vee F_3 \vee F_4 \vee F_7 \vee F_{24}$) shows that F_1 cannot be executed in parallel with F_2, F_3, F_4, F_7 and F_{24}. Before scheduling the edge, the implementation needs to evaluate if a flow path $F \in \mathcal{F}$ is sufficient to bind the edge, or if a collection of flow paths (composite route) is needed. For example, the edge $e_{6,8}$, modeling the transport of the output of $O_6(Mixer_2)$ (operation O_6 bound to component $Mixer_2$) to $O_8(Heater_1)$, can be directly bound to the flow path F_{15} (Table 4). The edge $e_{5,7}$ models the output of $O_5(Heater_1)$ being transported to $O_7(Mixer_3)$. However, there is no flow path $F \in \mathcal{F}$ that connects $Heater_1$ to $Mixer_3$. Therefore, a composite route (consisting of a collection of flow paths) needs to be generated. The edge $e_{5,7}$ is bound to the composite route (F_{30-1}, F_{26-1}) as shown in Fig. 5b.

The fluid transport latencies, c_F, associated with each flow path are also listed in Table 4. For calculating the latencies, we abstract away from absolute fluid volumes and utilize the concept of a unit fluid volume instead (captured by *metering* as explained in Section 2.1.1). Each fluidic I/O (input/output phase of a component) is characterized by a volume weight w_v, which is used to calculate the transport latency of a certain flow path when utilized for that specific fluidic I/O. Similarly, each component also has an associated capacity weight w_c, representing its volume capacity. For this example, we assume a volume weight of one for all fluidic I/Os. The capacity weight of all microfluidic components is assumed to be the same as its number of input phases, e.g., a mixer has two input phases, thus it has a capacity weight equal to two. Also, a fluid with volume weight one occupies a fixed channel length w_l on the chip. For the example, we assume this channel length to be equal to 10 mm.

The latencies for the flow paths have been calculated using a typical flow rate of 10 mm/s [9] and the chip dimensions of 5 mm between any two network vertices, N_i and N_j (termed as a segment), with $N_i, N_j \in \mathcal{N}$. For example, $F_1 = (In_1, S_1, Mixer_1)$ traverses two segments, i.e., In_1 to S_1 and S_1 to $Mixer_1$, thus a total channel length of 10 mm. With a flow rate of 10 mm/s, a fluid with volume weight one (occupying a total channel length of 10 mm) would have a total latency of 2 seconds from the time the fluid tip enters from In_1 till the fluid tail disappears into the mixer $Mixer_1$. During the scheduling phase, the storage requirement analysis needs to be performed as well. This means that after completion of an operation, a decision on whether the output fluid (analogous to the operand) should be moved to the storage reservoir or not, needs to be made.

3.3 Problem Formulation

The problem addressed in this paper can be formulated as follows: Given (1) a biochemical application modeled as a sequencing graph $\mathcal{G}$, (2) a biochip architecture modeled as a topology graph $\mathcal{A}$, and (3) a characterized component library $\mathcal{L}$, we are interested in synthesizing an implementation Ψ that minimizes the application completion time while satisfying the dependency, resource and routing constraints. Synthesizing an implementation $\Psi = <\mathcal{B}, \mathcal{X}>$ means deciding on (1) the binding $\mathcal{B}$ of each operation $O_i \in O$ to a component $M_j \in \mathcal{M}$, and each edge $e_{i,j} \in \mathcal{E}$ to a flow path $F_{i,j} \in \mathcal{F}$ (or to a composite flow path generated by the implementation), and (2) the schedule $\mathcal{X}$ of the operations and the edges, which contains the start time t^{start} of each operation O_i and edge $e_{i,j}$ on its corresponding component and (composite) flow path.

4. LIST SCHEDULING-BASED SYNTHESIS

The problem can be considered equivalent to the resource constrained scheduling problem with non-uniform weights, which is NP-complete [18]. We use a heuristic approach to solve the problem in a computationally efficient manner. Fig. 6 shows the block diagram of the proposed design methodology. In this paper, we focus on the 'Synthesis' box. It takes the biochemical application model and the flow layer models of the biochip architecture and the biochip components as input. As output, it produces the implementation Ψ which, together with the control layer models of the flow path set, $\mathcal{F}$, and the on-chip components, is used to generate the control sequence for executing the bioassay on the specified biochip.

The requirement of scheduling the routing together with the task operations, while satisfying the routing constraints, makes the problem analogous to the communication contention aware scheduling in parallel computing systems. Hence, we utilize the well-known *List Scheduling Algorithm* (LS) [11] and extend it with contention aware-

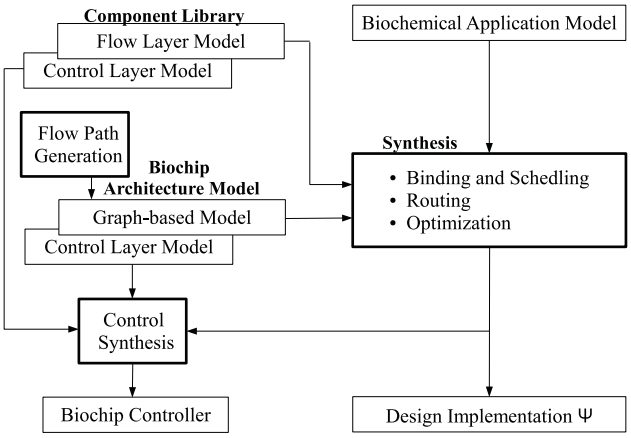


Figure 6: Design Methodology

BiochipSynthesis($\mathcal{G}$, $\mathcal{A}$, $\mathcal{L}$)

```
1  Initialize <B, B^o, X> to ϕ
2  while <all operations and edges are not scheduled> do
3     // Phase I: Bind Operations and Edges
4     while <all possible ready operations are not bound> do
5        B^o = BindOperations(G, A)
6        B^o = BindStorage(B^o, B, X)
7        B^o = GenRouteAndBindEdges(B^o, A, L)
8     end while
9     B = Record(B^o)
10    // Phase II: Schedule Operations and Edges
11    while <an operation event does not occur> do
12       X = ScheduleOperationsAndEdges(B, X, A)
13       Advance time to next event
14    end while
15 end while
16 return Ψ = <B, X>
```

Figure 7: Synthesis algorithm for flow-based biochips

ness [14] by also scheduling the edges ($\mathcal{E}$ in $\mathcal{G}$) onto the communication channels during the synthesis process.

The schedule shown in Fig. 5b has been generated using our LS-based synthesis algorithm shown in Fig. 7. The operations of the biochemical application are topologically sorted based on the dependency constraints. At each control step, operations are evaluated and the ready ones are found (the operations whose predecessor operations have been completed). Each new control step marks an operation event generation, with the operation event being defined as the completion of a scheduled operation. The list of ready operations is prioritized using the *urgency criteria*. The urgency of an operation is specified by the length of the longest path from the operation to the sink, i.e., summing up the execution weights of the vertices and the latency times for the edges. An average latency of 3 s is considered per edge based on the biochip architecture given in Fig. 4. An average is used since it is unclear on which flow paths the edges would be bound, unless binding of all operations has been completed (defining the source and destination component for each edge). The urgency value for O_1 in Fig. 5a is 25. If the number of ready operations exceeds the number of available resources, the most urgent operations (having higher urgency value) are scheduled and the remaining ones are deferred.

We perform the implementation synthesis in two phases. In Phase-I, we start off by binding the ready operations to the available resources (line 5). The algorithm tries all possible bindings and chooses the one that produces the shortest completion time for that operation. For example after control step 5, both $Mixer_1$ and $Mixer_3$ are available as the mixing operation O_7 is released, i.e., both its predecessor operations O_2 and O_5 have been completed. As shown in Fig. 5b, O_2 was bound to $Mixer_3$ and its output is still inside the mixer unit when O_7 is released (i.e., its output has not been moved to the storage unit). The algorithm binds O_7 to $Mixer_3$ preventing the routing delay that would have occurred had O_7 been bound to $Mixer_1$.

Next, we evaluate if a reservoir is required to store the output of the operations that finished in the previous control step (line 6) and if so, bind it to a particular reservoir. A storage reservoir is utilized only (1) if the component to which the previous operation was bound is needed for performing another operation and (2) the successor of the previous operation is not scheduled during the current control step. For example in Fig 5b, O_4 (bound to $Mixer_3$) was completed in control step 1. Its successor (O_6) is not scheduled in control step 2 and $Mixer_3$ is needed to perform O_2. Based on the above given criteria, output of O_4 is bound to the storage reservoir Res_1 at the start of control step 2.

Next we generate the routes (single flow path or composite route) for performing these operations and bind the edges to the generated routes (line 7). We start by fetching the phase information of the components (e.g., $Mixer_3$ for O_4) from the library $\mathcal{L}$ (Table 2) and bind the I/O phases (Ip1, Ip2, Op1, Op2) to the corresponding generated routes (e.g., F_3 for Ip1 of $Mixer_3$). If a route is not found, the operation is deferred. The algorithm jumps back to line 5 in order to modify the binding for the current control step ($\mathcal{B}^o$), by binding a low priority operation that was earlier deferred because of lack of resources. Once the binding has been finalized, it is recorded into the binding information $\mathcal{B}$ (line 9). More details about route generation are given in Section 4.1.

In Phase-II (lines 11–14), we generate the schedule for the operations and the edges bound in Phase-I. Starting from the input edges associated with the operation of the highest priority (based on the urgency criteria, O_4 in this case), we start scheduling the edges one by one (here the first one is F_3). In order to reduce the schedule length, we try to schedule as many bound edges as possible in parallel. All the ready edges that do not violate the routing constraints (listed in Table 4) can be utilized in parallel. An edge is considered a ready edge if it does not violate any inter- or intra-operation dependency constraints. The inter-operation dependency constraints are given by the application model $\mathcal{G}$ (e.g., O_3 and O_4 need to complete before O_6). The intra-operation dependency constraints means that the inputs of an operation and the operation itself need to be completed before its outputs can be issued (e.g., F_3, F_6 and the mixing operation in $Mixer_3$ need to complete before F_{22-1} and F_{19} can be scheduled). Multiple inputs for the same operation (F_3, F_6 for $Mixer_3$) are independent of each other and can be parallelized if the routing constraints permit. The same is applicable to the multiple outputs.

Fig. 8 shows the schedule for the first control step (Fig. 5b shows the complete schedule). Since O_4 (bound on $Mixer_3$) has the highest priority, F_3 is the first bound edge that is scheduled. None of the other edges (bound to flow paths) in the ready set $<F_6, F_2, F_5, F_1, F_4>$ can be scheduled in parallel with F_3 because of the routing constraints given in Table 4, thus F_3 is scheduled alone as shown in Fig. 8. Once all possible edges and operations are scheduled (only F_3 in this case), we advance time to the next event (operation event marking completion of an operation, or an edge event representing completion of an edge execution) (line 13). An operation event triggers a new control step and the algorithm switches back to Phase-I, whereas an edge event means that the next edge needs to be scheduled. Completion of

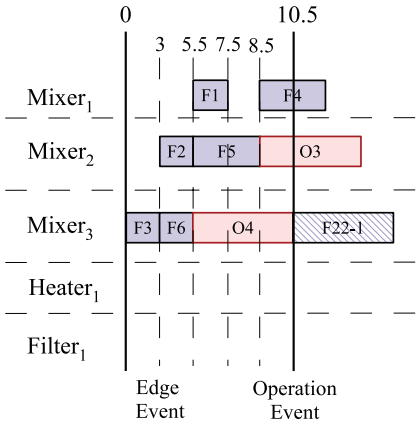


Figure 8: Edge and operation events

F_3 thus triggers an edge event. We schedule F_6 (next edge in the highest priority operation O_4) and try to optimize the schedule again (F_2 can be scheduled in parallel with F_6). The operations are scheduled as soon as all their input edges have been executed (e.g., O_4 after F_3 and F_6). When an operation finishes (e.g., O_4 on $Mixer_3$ in control step 1) it triggers an operation event and the algorithm jumps back to Phase-I.

The process is repeated until all operations/ edges are scheduled. The implementation Ψ, consisting of binding $\mathcal{B}$ and scheduling $\mathcal{X}$ information, is then returned (line 16).

4.1 Route Generation

GenRouteAndBindEdges (line 7) is used to generate the fluid route from the selected source to the selected destination (e.g., from In_1 to $Mixer_1$). If a flow path exists, then it binds the edge to that flow path and returns the binding information. If the selected operation has the selected source and destination such that no flow path exists to route the fluid from the source to the destination (e.g., $Heater_1 \rightarrow Mixer_3$, control step 5–7 in Fig. 5b), then the algorithm searches for a composite route, i.e., a route linking the desired source and destination using more than one flow path.

If multiple composite routes exist, the shortest one (in terms of cumulative latencies) is selected. For ($Heater_1 \rightarrow Mixer_3$), the composite route ($Heater_1 \rightarrow Res_1$, $Res_1 \rightarrow Mixer_3$) is selected. Note that this requires all intermediate destinations (Res_1 — reservoir 1 in $Storage - 8$ in the current case) to be available.

If no direct route is available and no composite route can be specified as well (e.g., intermediate destinations not available), then the operation is deferred. Phase-I (lines 4–8) is repeated again to see if any of the other low priority operations (that were not scheduled earlier because of resource constraints) can now be scheduled instead. Once all possible operations and edges are bound, the algorithm switches to Phase-II.

4.2 Optimization

We try to reduce the schedule length by considering the impact of fluid transport latencies on the routing constraints. As discussed in Section 2, the flow paths that share vertices cannot be utilized in parallel. This also covers the shared-resource constraints, i.e., a resource needs to be emptied before it can be reused. For example, as seen in Table 4, input flow paths of component $Heater_1$ (F_{10}, F_{15}, F_{20}, F_{27}) cannot be executed before or in parallel with its output flow paths (F_{30}, F_{31}), since $Heater_1$ needs to be emptied (output of the previous operation) before it can be reused. However, considering the flow-rate (10 mm/s) and the fluidic unit volume (10 mm of the channel length), we can see that $Heater_1$ would be emptied within 1 s, i.e., after 1 s of the output flow path activation the fluid would be out of $Heater_1$ and traversing the segment $Heater_1$ to S_9. None of the input flow paths of $Heater_1$ have a latency of less than 1 s. This means that the input and output flow paths of the heater can in fact be utilized in parallel with each other. Same applies to all other resources in the current chip as well, i.e., input flow paths of a resource can be utilized in parallel with (but not before) the output flow paths of the same resource. The schedule in Fig. 5b considers this in order to reduce the schedule length, e.g., for $Mixer_3$ (between control step 1 and 2) F_3 and F_{22-1} are executed in parallel.

Table 5: Experimental Results

Appl.	I/p Port	O/p Port	Mixer	Heater	Filter	Detector	Storage Units	$\delta_{\mathcal{G}}$
PCR	2	2	2	NR	NR	NR	3	42 s
	3	3	3	NR	NR	NR	1	40.5 s
	4	4	4	NR	NR	NR	2	34 s
IVD	2	2	2	NR	NR	2	0	63 s
	6	6	6	NR	NR	6	0	21 s
SB	2	2	3	1	1	NR	1	56 s
	4	8	7	2	1	NR	0	29.5 s

5. EXPERIMENTAL EVALUATION

We evaluate our proposed approach by synthesizing two real life assays and a synthetic benchmark onto different biochip architectures. The algorithm was implemented in C++, running on Lenovo T400s ThinkPad with Core 2 Duo Processors at 2.53 GHz and 4 GB of RAM.

Table 5 shows our experimental results. Columns 2-7 present the details of the architectures considered, in terms of input/ output ports, number of mixers, heaters, filters, detectors and storage units, respectively. The term NR in the table means that the component is not required for this application. The last column presents the completion time $\delta_{\mathcal{G}}$, in seconds, on the particular architecture.

The first real-life assay is the PCR (polymerase chain reaction) mixing stage that has 9 mixing operations and is used in DNA amplification [15]. We synthesize the assay on three different biochip architectures varying the number of I/O ports and mixer units. Each mixing operation is considered to have an execution time of 4 s on the mixing units used in the biochips. As given in Table 5, increasing the number of mixers and the I/O ports directly influences the application completion time $\delta_{\mathcal{G}}$ bringing it down to 34 s in the third architecture.

Multiplexed IVD (in-vitro diagnostics) has a total of 24 operations and is used to test different fluid samples from the human body [15]. As given in Table 5, IVD is synthesized onto two different biochip architectures. Each mixing operation is considered to have an execution time of 4 s and the detection operation 7 s. As we can see, increasing the number of mixers and detectors that can work in parallel, from 2 to 6, brings down $\delta_{\mathcal{G}}$ from 63 s to 21 s.

The example application given in Fig. 5a is used as a synthetic benchmark here and takes 56 s and 29.5 s on two different architectures. All of the experiments presented in this section took less than 1 s of run time to complete.

Varying the number of resources directly influences the chip area and also the schedule length. Chip area is an important parameter for the chips that need to be placed in small chambers, e.g., under microscopes for detection. Our methodology captures the design at the top level and can be utilized by the designer to evaluate their

Table 6: Results for Synthetic Benchmarks

Nodes	I/p Port	O/p Port	Mixer	Heater	Filter	Detector	$\delta_{\mathcal{G}}$
10	2	2	7	2	1	NR	31 s
	1	1	7	2	1	NR	31 s
20	2	2	7	2	1	NR	38 s
	1	1	7	2	1	NR	47 s
30	2	2	16	6	3	5	55 s
	17	19	16	6	3	5	54 s
40	2	2	17	10	10	3	69.5 s
	18	27	17	10	10	3	60 s

designed architectures and make design decisions at an early stage, minimizing the design cycle time and associated cost.

In a final set of experiments we have evaluated our proposed method using a set of four different synthetic benchmarks. The benchmark applications are composed of 10 up to 40 operations. Table 6 shows the details of the architectures considered and the respective application completion times achieved. No storage units were utilized by the applications in these experiments.

As shown in Table 6, we only vary the number of I/O ports in the considered architectures and note the impact on the application completion time $\delta_{\mathcal{G}}$. For the 10 node application, reducing the number of I/O ports from 2 to 1 has no influence on $\delta_{\mathcal{G}}$. However, when the same architecture is used for a different 20 node application benchmark, the completion time increases by approximately 23 %, going from 38 s to 47 s. For the 30 and 40 node applications, we first consider only 2 input and 2 output ports. And in the next architecture, we consider the optimal number of chip I/O ports, i.e., providing all the inputs required from off-chip reservoirs in parallel. We get varying results. For the 30 node application, increasing the I/O ports to the optimal number produces a gain of approximately 2 % in the completion time (55 s to 54 s), where as, for the 40 node application, the gain is slightly more significant, around 16 % (69.5 s to 60 s).

There are many factors that influence the application completion time, e.g., component interconnection scheme, application operation types and their sequence, operation execution times. The models presented in this paper can be used to further explore the relationships between these parameters, resulting in aiding design of optimal application-specific biochip architectures.

6. CONCLUSIONS

In this paper we have presented a system-level modeling and synthesis approach for flow-based microfluidic biochips. We have proposed a topology graph-based model to capture the biochip architecture and use a sequencing graph to model the biochemical application. The proposed approach synthesizes a biochemical application onto the specified biochip architecture with the application completion time minimization as the target objective. The synthesis process involves performing binding and scheduling of operations together with the routing (contention aware edge scheduling), while satisfying the dependency and resource constraints. Two real-life case studies and a set of four synthetic benchmarks have been synthesized on different architectures for validating the proposed approach. To the best of our knowledge, this is the first time such a system-level framework has been proposed for this type of chips. The proposed approach is expected to reduce human effort, enabling designers to take early design decisions by being able to evaluate their proposed architecture, minimizing the design cycle time and also facilitating programmability and automation.

7. ACKNOWLEDGMENTS

This work was supported by the Danish Agency for Science, Technology and Innovation, Grant No. 2106-08-0018 “ProCell”.

8. REFERENCES

[1] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson. Automatic volume management for programmable microfluidics. In *Proceedings of the 2008 ACM SIGPLAN conference*, 2008.

[2] K. Chakrabarty and T. Xu. *Digital Microfluidic Biochips: Design Automation and Optimization*. CRC Press, FL, 2010.

[3] K. Chakrabarty and J. Zeng. Design automation for microfluidics-based biochips. *Journal on Emerging Technologies in Computing Systems*, 1(3):186–223, 2005.

[4] H. Chou, M. Unger, and S. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, 3, 2001.

[5] C. Fang, Y. Wang, N. T. Vu, W.-Y. Lin, Y.-T. Hsieh, L. Rubbi, M. E. Phelps, M. Mueschen, 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*, 70(21), 2010.

[6] J. W. Hong and S. R. Quake. Integrated nanoliter systems. *Nature Biotechnology*, 21:1179–1183, 2003.

[7] I. Klammer, A. Buchenauer, H. Fassbender, R. Schlierf, G. Dura, W. Mokwa, and U. Schnakenberg. Numerical analysis and characterization of bionic valves for microfluidic pdms-based systems. *Journal of Micromechanics and Microengineering*, 17(7):S122–S127, 2007.

[8] E. T. Lagally, C. A. Emrich, and R. A. Mathies. Fully integrated pcr-capillary electrophoresis microsystem for dna analysis. *Lab on a Chip*, 1(2):102–107, October 2001.

[9] Y. C. Lim, A. Z. Kouzani, and W. Duan. Lab-on-a-chip: a component view. *Journal of microsystems tech.*, 16(12), 2010.

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

[11] G. D. Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, New York, 1994.

[12] J. M. Perkel. Microfluidics - bringing new things to life science. *Science*, November 2008.

[13] J. Siegrist, M. Amasia, N. Singh, D. Banerjee, and M. Madou. Numerical modeling and experimental validation of uniform microchamber filling in centrifugal microfluidics. *Lab Chip*, 10:876–886, 2010.

[14] O. Sinnen. *Task Scheduling for Parallel Systems*. John Wiley & Sons Inc., Hoboken, New Jersey, 2007.

[15] F. Su and K. Chakrabarty. Benchmarks for digital microfluidic biochip design and synthesis. *Technical Report, Duke University*, 2006.

[16] W. B. Thies. Programmable microfluidics. *presented at Stanford University*, October 2007.

[17] T. Thorsen, S. J. Maerki, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.

[18] D. Ullman. Np-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

[19] J. P. Urbanski, W. Thies, C. Rhodes, S. Amarasinghe, and T. Thorsen. Digital microfluidics using soft lithography. *Lab Chip*, 6:96–104, 2006.

[20] G. M. Whitesides. The origins and the future of microfluidics. *Nature*, 442:368–373, July 2006.