Fast Architecture-Level Synthesis of Fault-Tolerant Flow-Based Microfluidic Biochips

Wei-Lun Huang[†], Ankur Gupta[‡], Sudip Roy[‡], Tsung-Yi Ho[†] and Paul Pop[§]
[†]Dept. of Computer Science, National Tsing Hua University, Taiwan
[‡]CoDA Laboratory, Dept. of Computer Science & Engineering, Indian Institute of Technology Roorkee, India
[§]DTU Compute, Technical University of Denmark, Denmark
Email: love_i_suppor@yahoo.com.tw, ankursynon@gmail.com, sudiproy.fcs@iitr.ac.in,
tyho@cs.nthu.edu.tw, paupo@dtu.dk

***Abstract*—Microfluidic-based lab-on-a-chips have emerged as a popular technology for implementation of different biochemical test protocols used in medical diagnostics. However, in the manufacturing process or during operation of such chips, some faults may occur that leads to damage of the chip, which in turn results in wastage of expensive reagent fluids. In order to make the chip fault-tolerant, the state-of-the-art technique adopts simulated annealing (SA) based approach to synthesize a fault-tolerant architecture. However, the SA method is time consuming and non-deterministic with over-simplified model that usually derive sub-optimal results. Thus, we propose a progressive optimization procedure for the synthesis of fault-tolerant flow-based microfluidic biochips. Simulation results demonstrate that proposed method is efficient compared to the state-of-the-art techniques and can provide effective solutions in 88% (on average) less CPU time compared to state-of-the-art technique over three benchmark bioprotocols.**

I. Introduction

Microfluidic-based biochips (also known as lab-on-a-chip) integrate different components necessary for biochemical analysis (bioprotocol implementation), e.g., mixers, storage units, filters, and detectors, that helps in reducing the size of biochip at macroscopic level. Technological advances in microfluidic industry offer several advantages over the conventional laboratory procedures. Some of them include lower cost due to reduced volume of fluid consumption, higher levels of system integration, less likelihood of error due to minimal human intervention, faster biochemical reactions, and higher system throughput [1]. As a result, non-traditional biomedical applications and markets are opening up fundamentally new direction of chip design techniques. For example, the annual US market for in-vitro diagnostics is estimated at $10 billion, and $15 billion diagnostic tests/year worldwide [2].

Flow-based microfluidic biochips have become an innovative technology for implementation of real-life bioprotocols. A flow-based microfluidic biochip contains microchannels, microvalves and micropumps for manipulation of biochemical fluids over a small-size chip of few square centimeter area. The fluid transportation is carried out by the microchannels with the help of microvalves. Micropumps help in generating the required continuous pressure that makes fluid movement possible. Flow-based microfluidic biochips with more than $25,000$ valves and about a million features are commercially available to run $9,216$ polymerase chain reactions (PCRs) in parallel [3]. Moreover, the number of mechanical valves per square inch for flow-based microfluidic biochips has grown exponentially, which is four times faster than the reflection of Moore's Law [4]. However, biochip industry is still in its infancy. Hence, the possibility of manufacturing defects during fabrication process and operational faults increases as the size and complexity of chip increases. The failure of on-chip bioassay implementation because of these defects or faults may lead to wastage of hard-to-obtain and expensive biochemical fluids. In safety-critical applications, such microfluidic biochips should be fault-tolerant to avoid any risk.

It is difficult to transform the original architecture to its fault-tolerant version effectively, as almost all of the components can be duplicated. We propose a fast fault-tolerant design strategy, called as progressive optimization procedure (*POP*), that produces a fault-tolerant architecture for the biochip. The idea of this work is based on the partial reconfiguration of faulty regions using redundant components or microchannels.

Main Results: In this paper, a new and fast architecture-level synthesis technique is discussed for fault-tolerant design of flow-based microfluidic biochips. Two main contributions are as follows. First, the proposed technique is computationally feasible, and the execution (CPU) time grows linearly, when the number of faults in the given architecture increases. Second, the proposed technique tries to optimize the chip area for a given architecture. Thus, lesser area is required to tolerate the same number of faults by the proposed technique compared to the state-of-the-art technique (as shown in Fig. 1). Hence, remaining chip area can be utilized to tolerate more number of faults in the chip.

The remainder of the paper is organized as follows. Section II presents the prior work. Section III describes the system model and fault model for flow-based microfluidic biochip architectures. Motivation and problem formulation are discussed in Section IV. The proposed technique is described in Section V. Simulation results are presented in Section VI and finally, conclusions are drawn in Section VII.

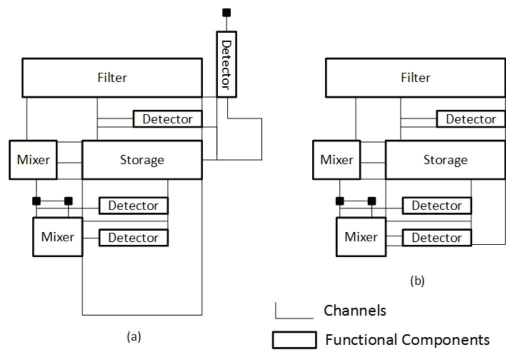

Fig. 1. Fault-tolerant architecture of an original flow-based microfluidic biochip architecture obtained by (a) previous work [5] and (b) proposed approach. A layout with lesser area is obtained in proposed approach.

II. Background and Prior Work

A. Background

Flow-based microfluidic biochips are constituted by microvalves and microchannels, which are manufactured from rubber-like elastomer (polydimethylsiloxane, PDMS). The schematic view of a typical flow-based microfluidic biochip architecture is shown in Fig. 2(a) having one mixer, one heater, one filter and one storage components. Microchannels are small tubes used for fluid transportation and microvalves (micro-mechanical valves) are used for restricting/permitting the flow. Typically, flow-based microfluidic biochip is divided into two layers: *flow layer* and *control layer*. Microvalves are controlled through the pressure source to permit/deny the fluid flow as shown in Fig. 2(b). When the pressure source is active, a fluid cannot flow through the valve (closed valve). In contrast, the fluid is permitted to flow through the valve unrestrained (open valve). The direction of fluid movement is also controlled by microvalves and microchannels.

Fig.2(c) shows an example of a mixer, composed of channels and nine valves, *v1* to *v9*. The valve-sets $\{v1,v2,v3\}$ *and* $\{v7,v8,v9\}$ act as control valves, which permit fluids to enter and leave the mixer, and the valve-set $\{v4,v5,v6\}$ acts as pump valves, which helps in mixing of fluids. If two fluids need to be mixed, first $\{v1,v2,v4,v5,v6,v7\}$ are opened and $\{v3,v8,v9\}$ are closed. Then one fluid is pumped to the mixer. When the channel is full with one fluid, *v2* and *v7* are closed and the fluid stays in the upper half of the mixer. Similarly, another fluid can be filled at the lower half of the mixer. Finally, all the valves except *v1* and *v9* are opened and $\{v4,v5,v6\}$ are used for mixing.

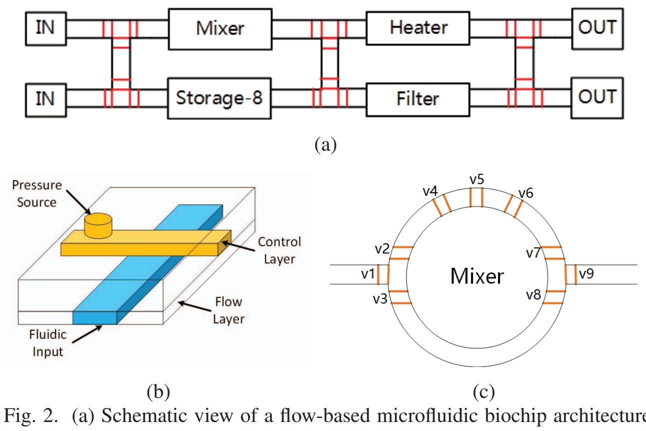

Fig. 2. (a) Schematic view of a flow-based microfluidic biochip architecture. (b) Microvalve at the junction of flow and control layers. (c) Microfluidic mixer.

B. Prior Work

Several fault-tolerant design strategies have been proposed for digital microfluidic biochips (DMFBs) in recent years [6–8]. DMFBs, like FPGA, contain grid of electrodes and are able to execute several biochemical processes concurrently. The fault issues can be resolved by having some redundant on-chip components. Although, flow-based microfluidic biochips, like ASIC, have no redundant components in their original architecture. To the best of our knowledge, there exists only one work that focuses on fault-tolerant mechanism for flow-based microfluidic biochips [5]. However, this work is based on greedy random adaptive method. This approach is time consuming and it generates non-deterministic result, and considers limited number of faults. Hu *et al.* [9, 10] presented an approach for automated functional testing of flow-based microfluidic biochips. Minhass *et al.* [11] presented the system modeling and synthesized the flow-based microfluidic biochips.

III. Architectural Modeling

A. Fault Model

The defect in flow-based microfluidic biochip can be either block or leak [10]. A fault model for a biochip architecture $\mathcal{A}$ is described as follows: $Z = (VF, CF, V_{max}, C_{max})$, where VF and CF are finite sets of all valve faults and channel faults, respectively. V_{max} and C_{max} specify the maximum number of valve and channel faults, respectively, those may simultaneously occur in the architecture $\mathcal{A}$. A valve fault is described as $VF(n,w,t)$, where n represents the faulty component, w is the affected valve inside the component n, and t denotes the type of valve fault. Similarly, a channel fault may occur either in component n or in channel $d_{i,j}$, denoted by $CF(n,t)$ / $CF(d_{i,j},t)$, where t is the type of the fault. Each architecture has its own fault model, which is generated by the experienced designer.

B. Fault Scenarios

An example architecture presented in Fig. 2(a) and the fault model $Z = (VF, CF, 2, 2)$ is listed in Table I. Thus, some possible fault scenarios are as follows: $\{CF_2\}$, $\{VF_1, CF_1\}$, $\{VF_1, VF_2, CF_1, CF_2\}$, $\{\phi\}$, etc. Each fault scenario is unique as it occurs only once. Considering all combinations of faults in Z, $|Z|$ number of different fault scenarios are possible. An architecture is considered fault-tolerant to Z, if an application is able to complete within its deadline d under all possible fault scenarios in Z.

TABLE I
FAULT MODEL FOR THE ARCHITECTURE IN FIG. 2(A).

Name	Component ($n \in N$) / Affected Valve (w)	Type
VF$_1$	Mixer / v4	open
VF$_2$	Switch s6 / v3	open
VF$_3$	Switch s2 / v3	open
VF$_4$	Switch s3 / v1	open
Name	**Component** ($n \in N$) / **Channel** ($d_{i,j} \in D$)	**Type**
CF$_1$	Heater	Block
CF$_2$	Mixer	Block
CF$_3$	Switch s4 $\rightarrow$ Filter	Leak
CF$_4$	Switch s5 $\rightarrow$ Switch s8	Block

C. Fault-Tolerant Components

Designers are given with a component library L, which includes both fault-tolerant (prefixed with "FT-") and original version of all the components as shown in Table II. In this paper, we assumed that fault-tolerant components can tolerate faults that occur in original components. However, these FT-components may increase the required chip area or execution time, i.e., the cost for the design. The components in L are built using valves, channels, heaters (actuators), and sensors (detectors).

A switch is shown in Fig. 3(a), which helps in fluid transportation. Fig. 3(b) shows the fault-tolerant version of a switch. A FT-switch has four additional valves compared to the original switch and these valves can tolerate faults caused by stuck-open valve faults and faulty valve is replaced by the redundant valve. However, if a valve is stuck-closed, the switch is still faulty and the fluid is restricted from leaving the switch to the connected channel. As a result, a complete new channel is required to tolerate the stuck segment, and the design cost will increase accordingly.

TABLE II
COMPONENT LIBRARY L

Component	Phases	Area	Valves
Mixer	Input1/Input2/Mix/Output1/Output2	30x30	9
FT-Mixer	Input1/Input2/Mix/Output1/Output2	30x30	10
Filter	Input/Filter/Output1/Output2	120x30	2
FT-Filter	Input/Filter/Output1/Output2	120x60	6
Detector	Input/Detector/Output	20x20	2
FT-Detector	Input/ Detector /Output	20x40	6
Separator	Input1/Input2/Separator/Output1/Output2	70x20	2
FT-Separator	Input1/Input2/Separator/Output1/Output2	70x40	6
Heater	Input/Heat/Output	40x15	2
FT-Heater	Input/Heat/Output	40x30	6
Metering	Input/Met/Output1/Output2	30x15	6
Multiplexer	Input/Output	30x10	2
Storage	Input/Storage/Output	90x30	28
FT-Storage	Input/Storage/Output	90x40	34

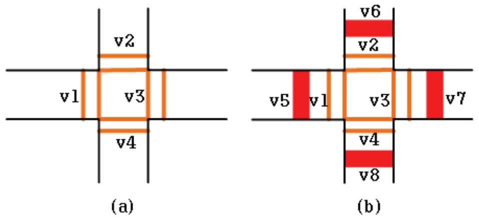

Fig. 3. Four-direction switch. (a) Regular design, and (b) fault-tolerant design.

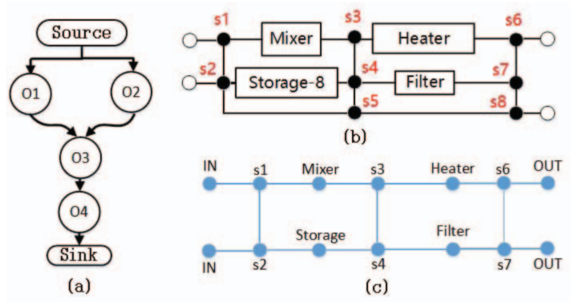

Fig. 4. A set of system model. (a) An example sequencing graph G, (b) an example architecture for implementing G in (a), and (c) an example topology graph A of (b).

D. System Model

A netlist of a flow-based microfluidic biochip architecture $\mathcal{A}$ includes locations, the link information (inputs and outputs), and other information of components and channels. A simple example of biochip architecture is shown in Fig. 4(b), which has two inputs, two outputs, and different components: mixer, heater, filter and storage space. The architecture of flow-based microfluidic biochip is described as a topology graph $A = (N, D)$ (Fig. 4(c)), where N represents set of vertices corresponds to components and D represents set of edges corresponds to channels. An application is modeled using a sequencing graph $G = (O, E)$, where O denotes the set of operations and E represents the set of directed edges indicating the dependency between corresponding operations in the assay (Fig. 4(a)). Each vertex and edge in graph G has its own weight, which denotes the execution time during the operation and movement, respectively. Applications have their own deadline d and $d > T_c$, where T_c represents the completion time of given application.

IV. MOTIVATION AND PROBLEM FORMULATION

A. Motivation

In recent years, the microfluidic biochip industry has grown much faster. Technological advancement makes it possible to accommodate thousands of components into few square centimeter size chip. The increasing complexity of chip creates possibility of manufacturing defects and operational faults that leads to failure in on-chip execution of biochemical application. An efficient approach is desired to make the architecture fault-tolerant so that longer experimental time can be avoided by re-executing the bioassay under fault detection.

B. Problem Formulation

We formulate the problem as follows.
Inputs: (1) Topology graph A for an architecture $\mathcal{A}$, (2) sequencing graph G of an application with deadline d, (3) a component library L, and (4) a fault model Z.
Output: A fault-tolerant architecture $\mathcal{A}^+$.
Objective: To minimize the cost of the fault-tolerant architecture including the area usage.
Subject to: (1) The fault-tolerant architecture can successfully execute the biochemical application within its deadline d, even

though the expected fault scenario occurs. (2) Each fault in the fault model Z can be tolerated.

V. Fault-Tolerant Architecture-Level Synthesis

The problem presented in the previous section is *NP-hard* [12], because each time when a redundant component is duplicated in the architecture $\mathcal{A}$, the entire empty space in the chip is used. Hence, it is difficult to enumerate the total number of possible paths. In order to find the minimum-cost paths, it is unnecessary to consider all possible paths. We adopt a progressive approach, which can find an effective solution for the fault scenario in a short time. The flowchart of the proposed synthesis technique, called as fault-tolerant architecture-level synthesis (FTAS), is shown in Fig. 5 and symbols with their descriptions are listed in Table III.

We propose a progressive algorithm with four stages for tolerating channel faults: (1) path finding; (2) constrained path finding; (3) duplication; and (4) detouring. If one stage is able to handle a fault, then no need to continue to execute the following stage(s). First, the objective of path finding stage is to build a shortcut, which is used to avoid the faulty channel usage. Therefore, if the fault is handled by this stage,

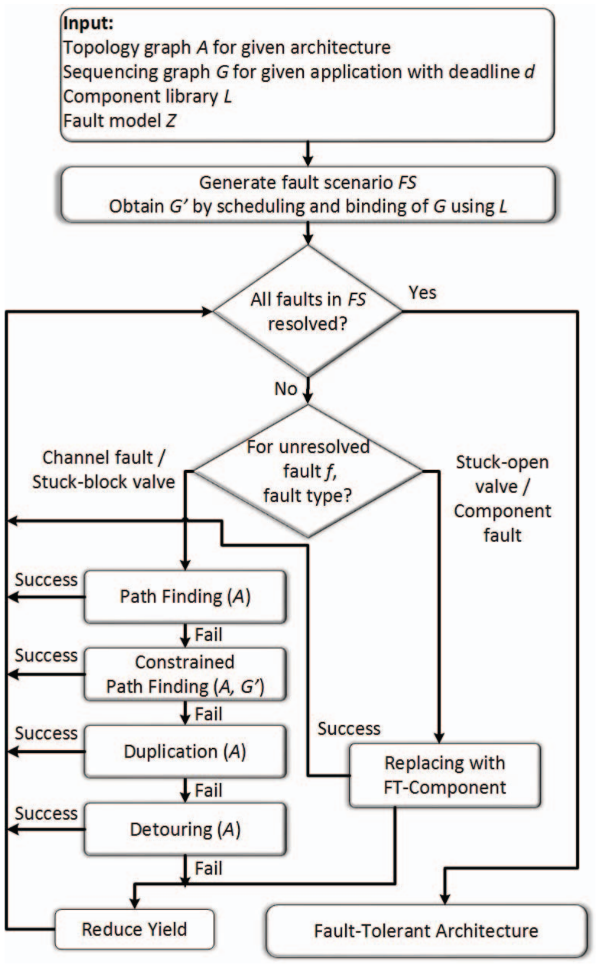
Input:
Topology graph A for given architecture
Sequencing graph G for given application with deadline d
Component library L
Fault model Z
Generate fault scenario FS
Obtain G' by scheduling and binding of G using L
All faults in FS resolved?
Yes
No
For unresolved fault f, fault type?
Channel fault / Stuck-block valve
Stuck-open valve / Component fault
Success
Path Finding (A)
Fail
Constrained Path Finding (A, G')
Duplication (A)
Detouring (A)
Replacing with FT-Component
Reduce Yield
Fault-Tolerant Architecture

Fig. 5. Flow-chart of *FTAS*.

TABLE III
List of Notations Used and Their Descriptions.

Symbol	Description	Symbol	Description
$\mathcal{A}$	Normal biochip architecture	$\mathcal{A}^+$	Fault-tolerant biochip architecture
VF	Valve fault	CF	Channel fault
V_{max}	Maximum number of VF	C_{max}	Maximum number of CF
N	Set of components	D	Set of channels
Z	Fault model	FS	Fault scenario
G	Sequencing graph of application	d	Deadline
L	Component library	FT-	Fault-tolerant version
Symbol	**Description**		
T_c	Completion time of an application using the architecture (with faults)		

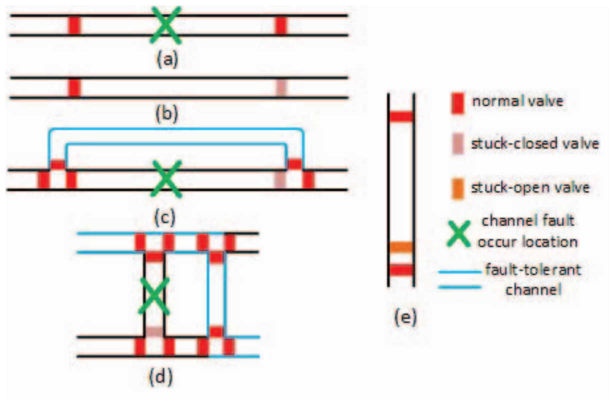

Fig. 6. Example of tolerating different value-faults. (a) A channel fault, (b) a stuck-closed type valve-fault, (c) tolerance of faults in (a) and (b) (a stuck-block valve can be tolerated by fault-tolerant mechanism of channel fault), (d) a redundant pathway shown in 'blue' color for stuck-closed valve-fault, and (e) a stuck-open type valve-fault can be tolerated by a redundant valve.

the completion time of application T_c will decrease. Second, the constrained path finding stage refers operation sequencing graph G and finds paths with a design cost of fault-tolerant architecture $\mathcal{A}^+$ lower than that of duplicating the redundant channels. This stage does not increase the completion time T_c (discussed in Subsection V-B). Third, the faulty channel is duplicated by the redundant channel. This stage may increase the completion time T_c, as explained in Fig. 6(c, d). If all these previous stages are not able to handle the fault, detouring is required to handle the fault that occurs in critical channel. However, the design cost of fault-tolerant architecture $\mathcal{A}^+$ and completion time T_c increase significantly.

The other type of fault is valve fault, which is comparatively easier to handle. If the type of valve fault is stuck-open, it means that the fluid can flow through channels, but the direction of flow cannot be controlled. In this case, the redundant valve is placed beside the faulty one as shown in Fig. 6(e). The other type of valve fault is stuck-closed, and it is equivalent to channel fault as shown in Fig. 6(a). Stuck-closed valve fault maintains the block state and fluid cannot flow through the neighbor channels as shown in Fig. 6(b). A channel fault and stuck-closed value fault are tolerated by placing a redundant channel as shown in Fig. 6(c).

A. Path Finding

The first stage is path finding that tries to find shortcuts, which give better solutions than duplicating a redundant channel. Fig. 7 shows a possible fault, which is predicted on the

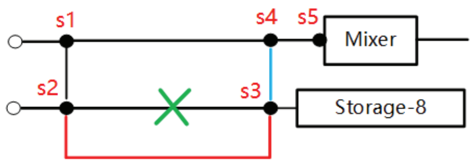

Fig. 7. An example of path finding phase. ‘Blue’ channel is a better choice than ‘red’ channel.

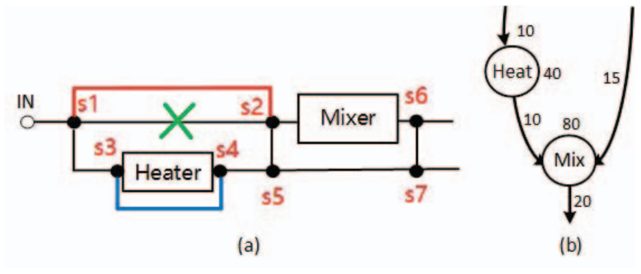

Fig. 8. (a) An example fault-tolerant flow-based microfluidic biochip architecture. (b) Schedule of operations for the architecture in (a) (numbers indicate the execution time units for each operation or transportation).

channel marked with ‘green’ cross. The channel (drawn in ‘red’) is a possible solution, which is a case of duplication. Sometimes, a shortcut of pathway can be built once the critical path is found from storage-8 to the mixer as shown in Fig. 7 (drawn in ‘blue’). This shortcut not only tolerates the fault but also reduces the completion time T_c. In order to find these kind of shortcut paths, a weighted directed graph G' is built from the topology graph A of architecture $\mathcal{A}$. The start and end components are source and sink in G', respectively. Each channel is represented by an edge and each possible switch is represented by a vertex (i.e., $s1$, $s2$, and $s4$ in Fig. 7). If an edge in A includes new channels or valves, unit cost weight is assigned to that edge. Otherwise, the weight of the edge is set to zero. However, several possible shortcuts with lower unit costs and completion time T_c are possible, and all of them are saved.

B. Constrained Path Finding

The second stage is constrained path finding that tries to find some replaceable paths from the sequencing graph G of application. For example shown in Fig. 8(a), the cross mark (in ‘green’) indicates the channel fault and the redundant channel (shown in ‘red’) is a simpler solution. Nevertheless, the application shown in Fig. 8(b), represents that the channel $s1$ to $s2$ is used to allow the fluid to flow from input to mixer. The critical path for completion time T_c is waiting for another fluid to the mixer after heating. Hence, another path (shown in ‘blue’) is a better choice to tolerate the channel fault from input to mixer. This path does not increase the completion time T_c and has a lower design cost as well.

C. Duplication

Duplication is copying the same (redundant) channel beside the faulty one. This stage is able to handle most of the channel faults. If a channel fault cannot be tolerated by this method,

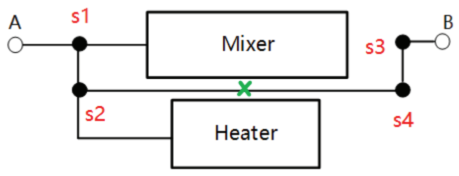

Fig. 9. An example case of detouring.

it is difficult to find any path with lower design cost. In such case, enough empty space around the targeted channel is not available. Hence, the solution will get a comparable design cost from the next stage that may take extra completion time T_c than using the original channel.

D. Detouring

The detouring stage tolerates the channel fault, if not handled by the previous stages. For example, Fig. 9 shows that the channel $s2$ to $s4$ is critical. If this channel is faulty, the entire architecture will be out of action. Nevertheless, there is no enough empty space around the channel to tolerate this fault. During this stage, the path is divided into two segments: one segment includes vertices A, $s1$, and $s2$ and the other segment includes B, $s3$, and $s4$. Next, multi-terminals net connected method (extended from Lee’s algorithm) is used to connect these two segments [13]. Due to this overhead, the design cost of the architecture $\mathcal{A}^+$ and completion time T_c will increase. If the completion time T_c exceeds the deadline d at the end of synthesis, the current solution is discarded. Hence, the channel fault cannot be tolerated under the given fault model for architecture $\mathcal{A}$.

E. Replacing with FT-Component

If there is a channel fault or a valve fault occurred in a functional component, then that component need to be replaced by the fault-tolerant version or another functional component of same specification is added to obtain the fault-tolerant architecture $\mathcal{A}^+$. Replacing a component with FT-component incurs less design cost and area in comparison to adding a new functional component. Usually, FT-component is larger in size than the non fault-tolerant version. Hence, replacing with FT-component may need to shift the affected component around the available empty space. This shifting is done in such a way that the components, which has minimal overlapping area with other components are selected. Then a ripple-like method is used to move the overlapped components.

F. Execution Time Estimation

In order to optimize the completion time T_c for the given application, an specific parameter is used to record the completion time T_c. This parameter is important to determine, if the application can be finished within the deadline d without using the scheduling and binding when a new fault-tolerant unit is added each time. This step is quite simple, but it can save more time for the fault-tolerant synthesis flow.

TABLE IV
COMPARATIVE RESULTS OF NUMBER OF COMPONENTS $|N|$, NUMBER OF MICROCHANNELS $|D|$ AND CPU TIME (IN *hh* : *mm* : *ss*) FOR THREE APPROACHES *SFS*, *GRASP* AND *POP*.

Name	$\mathcal{A}$				$\mathcal{A}^+_{SFS}$ [5], [14]			$\mathcal{A}^+_{GRASP}$ [5], [14]			$\mathcal{A}^+_{POP}$		
	$\lvert N\rvert$	$\lvert D\rvert$	$\lvert FS\rvert$	$\lvert FS^-\rvert$	$\lvert N\rvert$	$\lvert D\rvert$	CPU Time	$\lvert N\rvert$	$\lvert D\rvert$	CPU Time	$\lvert N\rvert$	$\lvert D\rvert$	CPU Time
SB1	15	17	121	100	20	27	06:23:54	15	20	01:22:36	15	27	00:18:11
PCR	14	16	77	50	18	25	16:05:57	14	17	02:23:52	14	28	00:19:09
IVD	52	78	841	100	57	92	$>$ 48hr	52	78	27:40:37	52	95	00:31:50

VI. SIMULATION RESULTS

We have implemented the proposed method progressive optimization procedure (*POP*) using C++ and a machine with Xeon 3.50GHz CPU and 128GB of memory for execution of algorithm code. Simulation is performed for three cases, one synthetic test case SB1, and two real-life test cases IVD (In-Vitro-Diagnostics) and PCR (Polymerase Chain Reaction) [15]. For each test case, we used the tools of [15] to generate a non-fault-tolerant architecture $\mathcal{A}$. Features of architecture $\mathcal{A}$ are presented in Table IV, where $|N|$ denotes the number of microfluidic components and $|D|$ represents the number of microchannels. We randomly generate a fault model Z, for each test case, based on the type of fault that may occur in the architecture $\mathcal{A}$. Consider that $|FS|$ be the total number of fault scenarios, which is generated from fault model Z. As in the previous work [5], FS^- is considered as the subset of fault scenarios FS, and $|FS^-|$ represents the total number of fault scenarios in the set FS^-. If the reduced fault scenarios $|FS^-|$ are not considered, then the computation (CPU) time will grow significantly. However, by considering only subset FS^- during synthesis most of the fault scenarios are covered. We compare our technique *POP* with the previous work [5] that introduced two fault-tolerant architectures $\mathcal{A}^+_{SFS}$ and $\mathcal{A}^+_{GRASP}$ generated from $|FS|$ and $|FS^-|$, respectively.

As shown in Table IV, *POP* is faster to generate the fault-tolerant architecture $\mathcal{A}^+_{POP}$ than both $\mathcal{A}^+_{SFS}$ and $\mathcal{A}^+_{GRASP}$. For example, on comparison of the average of execution (CPU) time of all three algorithms, we found that *POP* is $27.3\times$ and $> 100\times$ faster than $\mathcal{A}^+_{GRASP}$, respectively. Main reasons are that the previous work [5] iteratively runs the binding and scheduling on semi-finished architecture after every new redundant unit is considered. Moreover, simulation results show that *POP* is scalable that means with the increase in the architecture size and the size of the fault scenarios, the execution (CPU) time of *POP* increases linearly. The proposed technique *POP* uses the same number of components as the original architecture $\mathcal{A}$ that represents *POP* can tolerate all the faults occurred in components by using FT-components and the unit cost of $\mathcal{A}^+_{POP}$ is low. While the number of microchannels used considered, *POP* uses comparatively more microchannels than previous work [5] due to two reasons. First, not all the fault scenarios are considered by the previous method $\mathcal{A}^+_{GRASP}$ and some of the faults in FS may not be tolerated by $\mathcal{A}^+_{GRASP}$. Second, *POP* considers each microchannel and microvalve necessary in the original architecture $\mathcal{A}$, so performs fault-tolerant mechanism for every fault in the fault model Z. However, as long as the architecture is connected, each component has at least one microchannel that can operate correctly. The architecture can finish the application even with some faults, if unlimited deadline is assigned.

VII. CONCLUSIONS

In this paper, we present an efficient and effective method to synthesize fault-tolerant architecture for flow-based microfluidic biochips. Simulation results show that the proposed method is faster than the greedy random adaptive method of state-of-the-art. Moreover, the proposed approach is scalable even when the problem size is large. However, if the components can be rotated, it can save more space in the architecture, which can be studied as a future work. Another possible scope is to improve the ripple-like method for FT-components in the fault-tolerant architecture.

REFERENCES

[1] D. Mark, S. Haeberle, G. Roth, F. von Stetten, and R. Zengerle, "Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications," *Chemical Society Reviews*, vol. 39, no. 3, pp. 1153–1182, 2010.

[2] K. Chakrabarty and Y. Zhao, "Toward fault-tolerant and reconfigurable digital microfluidic biochips," in *Proceedings of IEEE Asia Symposium on Quality Electronic Design*, 2010, pp. 198–207.

[3] J. M. Perkel, "Life science technologies: microfluidics - bringing new things to life science," *Science*, vol. 322, no. 5903, pp. 975–977, 2008.

[4] J. W. Hong and S. R. Quake, "Integrated nanoliter systems," *Nature biotechnology*, vol. 21, no. 10, pp. 1179–1183, 2003.

[5] M. C. Eskesen, P. Pop, and S. Potluri, "Architecture synthesis for cost-constrained fault-tolerant flow-based biochips," in *Proceedings of IEEE Design Automation and Test in Europ Exhibition*, 2016, pp. 618–623.

[6] F. Su and K. Chakrabarty, "Yield enhancement of reconfigurable microfluidics-based biochips using interstitial redundancy," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 2, no. 2, pp. 104–128, 2006.

[7] Y.-H. Chen, C.-L. Hsu, L.-C. Tsai, T.-W. Huang, and T.-Y. Ho, "A reliability-oriented placement algorithm for reconfigurable digital microfluidic biochips using 3-d deferred decision making technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1151–1162, 2013.

[8] F. Su and K. Chakrabarty, "Design of fault-tolerant and dynamically-reconfigurable microfluidic biochips," in *Proceedings of IEEE Design Automation and Test in Europe*, 2005, pp. 1202–1207.

[9] T.-Y. Ho, K. Chakrabarty, and K. Hu, "Testing of flow-based microfluidic biochips," in *Proceedings of IEEE VLSI Test Symposium*, 2013, pp. 1–6.

[10] K. Hu, F. Yu, T.-Y. Ho, and K. Chakrabarty, "Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1463–1475, 2014.

[11] W. H. Minhass, P. Pop, and J. Madsen, "System-level modeling and synthesis of flow-based microfluidic biochips," in *Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the 14th International Conference on*. IEEE, 2011, pp. 225–233.

[12] T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*, 2007.

[13] K. Lampaert, G. Gielen, and W. Sansen, *Analog layout generation for performance and manufacturability*. Springer Science & Business Media, 2013, vol. 501.

[14] M. C. Eskesen, "Fault-tolerant architecture design for flow-based biochips," *Master Thesis, Technical University of Denmark*, p. 100, 2015.

[15] W. H. Minhass, P. Pop, and J. Madsen, "System-level modeling and synthesis techniques for flow-based microfluidic very large scale integration biochips," Ph.D. dissertation, Technical University of Denmark Tekniske Universitet, 2012.