Synthesis of Application-Specific Fault-Tolerant Digital Microfluidic Biochip Architectures

Mirela Alistar, Paul Pop, and Jan Madsen

***Abstract*—Digital microfluidic biochips (DMBs) are microfluidic devices that manipulate droplets on an array of electrodes. Microfluidic operations, such as transport, mixing, and split, are performed on the electrode array to perform a biochemical application. All previous work assumes that the DMB architecture is given and most approaches consider a rectangular shape for the electrode array. However, nonrectangular application-specific architectures are common in practice. Hence, in this paper, we propose an approach to the synthesis of application-specific architectures, such that the cost of the architecture is minimized and the timing constraints of the biochemical application are satisfied. DMBs can be affected by permanent faults, which may lead to the failure of the biochemical application. Our approach introduces redundant electrodes to synthesize fault-tolerant architectures aiming at increasing the yield of DMBs. We have used a tabu search metaheuristic for this architecture synthesis problem. We have proposed a technique to evaluate the architecture alternatives visited during the search, in terms of their impact on the timing constraints of the application. The proposed architecture synthesis approach has been evaluated using several benchmarks.**

***Index Terms*—Architecture synthesis, digital microfluidic biochips, fault-tolerance.**

I. Introduction

With the introduction at the beginning of 1990s of microfluidic components such as microvalves and micropumps, it was possible to realize "micro total analysis systems," also called "lab-on-a-chip" or "biochips," for the automation, miniaturization, and integration of complex biochemical protocols. Microfluidic platforms are used in many application areas, such as, *in vitro* diagnostics (point-of-care, self-testing), drug discovery, biotech, and ecology [1], [2].

Microfluidic platforms can be classified according to the liquid propulsion principle used for operation, e.g., capillary, pressure driven, centrifugal, electrokinetic, or acoustic. In this paper, we are interested in microfluidic platforms which manipulate the liquids as droplets, using electrokinetics, i.e., electrowetting-on-dielectric (EWOD) [3]. We call such platforms digital microfluidic biochips (DMBs). DMBs are able to perform operations such as dispensing, transport, mixing, split, dilution, and detection using droplets [1].

Manuscript received July 15, 2015; revised October 31, 2015; accepted January 11, 2016. Date of publication February 10, 2016; date of current version April 19, 2016. This paper was recommended by Associate Editor S. Hu and A. Zomaya.

The authors are with the Technical University of Denmark, Kongens Lyngby 2880, Denmark (e-mail: mali@dtu.dk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2528498

A. Related Work

The architecture of a DMB consists of physical components, such as electrodes, detectors, heaters, reservoirs for dispensing, and waste. Previous work assumes that the physical architecture of a DMB is given. Most researchers use general-purpose architectures, which have a rectangular shape [Fig. 1(a)]. However, in practice, nonregular application-specific architectures (Fig. 5) are more common because they can significantly reduce the cost by including only the components that are necessary for the execution of the application. For example, application-specific architectures have been proposed for disease screening in newborns [4] and for diagnostics on human physiological fluids [2]. Application-specific architectures are designed manually, which is an expensive time-consuming process. Hence, there is an imperative need for methodologies to automate the design of application-specific DMBs.

Yield is a big concern for biochips, hence, the focus on fabrication methodologies to increase the yield of DMBs, e.g., from a very low 30% to 90% [5]. Permanent faults (e.g., due to abnormal layer deposition, short between two adjacent electrodes, see [6], [7] for more details) are usually introduced during fabrication and may lead to the failure of the biochemical application. After fabrication, the biochips are tested and if permanent faults are detected, the biochip is discarded unless the applications can be reconfigured to avoid them [8]. To increase the yield, which is very important for the market success of DMBs, the design of DMBs has to consider possible defects that can be introduced during the fabrication process.

Researchers have used the term "synthesis" to denote the tasks that determine the "electrode actuation sequence," which controls the movement of droplets on the biochip. We will call these synthesis tasks compilation, to distinguish it from the architecture synthesis proposed in this paper. The following are the main design tasks that have been addressed [1].

1) First, the biochip architecture is synthesized—process that is currently done manually.
2) After the architecture is synthesized, the biochip is fabricated [3] and tested to determine the location of the permanent faults [6].
3) Next, the biochemical application has to be compiled to determine the electrode actuation sequence needed to run the biochemical application. In case the compilation is not successful (e.g., the biochip cannot be reconfigured to run the application), the biochip is discarded.

In this paper, we address the architecture synthesis problem, that is deciding on a physical fault-tolerant biochip architecture of minimum cost. Researchers have so far only considered varying the dimensions of purely rectangular general-purpose architectures or have addressed aspects such as minimizing the number of pins used to control the electrodes [9], which is orthogonal to our problem. Also, previous work has addressed the issue of fault-tolerance only in the context of rectangular architectures, by proposing post-fabrication compilation approaches. Su and Chakrabarty [5] proposed a compilation approach that uses the available electrodes to replace the faulty electrodes. The approach has been tested for one application using biochips architectures with hexagonal electrodes. A more general compilation is proposed in [8], where the droplets routes are reconfigured to avoid the faulty electrodes. In [10], we have proposed a first approach to the architecture synthesis, which decides the allocation of physical components, their placement and interconnection, such that the cost is minimized and the application satisfies the timing constraints even in the presence of permanent faults.

Compilation is used inside the architecture synthesis, for the evaluation of the alternative architectures generated during the solution space exploration. The compilation process is an NP-complete problem for which several approaches have been proposed. Maftei *et al.* [11] proposed an integer linear programming formulation that derived optimal solutions for small applications. Near-optimal results in terms of application completion time were obtained by compilation implementations based on search metaheuristics such as simulated annealing (SA) [1] and tabu search (TS) [12]. List scheduling (LS)-based compilations were proposed in [13]–[15]. These compilations are faster, and thus, can be used to quickly evaluate an alternative architecture during architecture synthesis.

In order to be compiled, biochemical applications are modeled using process graph models, where each node is an operation, and each edge represents a dependency [1]. Next, the necessary modules for the execution of the operations are allocated from a module library. As soon as the binding of operations to the allocated modules is decided, the scheduling algorithm determines the duration for each bioassay operation, subject to resource constraints and precedence constraints imposed by the application. Several approaches have been proposed for binding and scheduling, which is an NP-complete problem [13], [14], [16]. Next, the placement [17] of each module on the microfluidic array and the routing [18], [19] of droplets have to be determined.

In the context of application-specific architectures, which have an irregular layout, the placement problem becomes more challenging. In related literature, several placement strategies have been proposed for rectangular architectures. In [20], an SA-based method is used to determine the placement of the operations on the biochip. A unified compilation and module placement, based on parallel recombinative SA, was proposed in [21].

Better results were obtained using a T-Tree algorithm for placement [17] or using a fast-template placement [22] integrated in a TS-based compilation [12]. A placement approach that minimizes droplet routing, when deciding the locations of the modules, is considered in [23]. Placement strategies based on virtual topology [16] were proposed for fast compilation approaches.

All mentioned approaches consider placement of rectangular modules, which do not take advantage of the irregular area of an application-specific biochip. A placement strategy for modules of nonrectangular shapes is proposed in [12]. However, Maftei *et al.* [12] used a black-box approach, i.e., the whole module area is occupied during the execution of the operations, blocking the corresponding electrodes from being used for other operations. An alternative is the routing-based approach from [24], which allows the droplets to move freely on the biochip until the operation is completed. However, in case of contamination, the routing-based strategy requires a lot of washing, which slows considerably the execution of the bioassay and can lead to routing deadlocks.

In [25], we have proposed the first placement algorithm for application-specific architectures. Our placement strategy starts from an application-specific architecture, given as input, and determines circular-route modules (CRMs) of any shape, on which operations execute so that the biochip area is effectively used.

B. Contributions

Starting from a biochemical application and a library of physical components, our architecture synthesis (see Algorithm 3) decides a physical biochip architecture that minimizes the cost and executes the application within its specified deadline even in the case of k permanent faults. Our proposed approach starts from an initial architecture solution and uses a TS metaheuristics to search the solution space and generate new architectures. Each architecture alternative visited by TS has to be evaluated in terms of cost, fault-tolerance, and timing constraints of the application. We propose an LS-based compilation that determines the completion time of the considered application when it is executed on the architecture under evaluation.

Our LS-based compilation considers maximum k permanent faults and estimates their impact on the application completion time. The difficulty of obtaining a fault-tolerant architecture lays in not knowing the exact position of the faults when the architecture synthesis is performed, that is, before the biochip fabrication and testing. Alistar *et al.* [10] determined the worst-case schedule length in case of maximum k permanent faults, by considering that each operation in the application suffers from k faulty electrodes. The approach from [10] is pessimistic, and it results in rejecting potentially good architecture solutions. Instead, we propose an estimation method for the application execution time, which is less pessimistic than the worst-case values. Our estimation method is faster and thus more suitable to be used inside the TS-based architecture synthesis.

Our solution in [10] to the architecture synthesis problem, considered rectangular modules with empty insides and varying border thickness. Although such modules are suitable for placement on application-specific biochips, their rectangular shape does not permit an effective use of the area on the biochip, due to its irregular layout. Hence, in this paper we use CRMs for operation execution.

In [25], we have proposed an algorithm that builds a library $\mathcal{L}$ of CRMs for a given application-specific architecture $\mathcal{A}$. Faults are not considered by [25], as the focus is on determining CRMs that will use effectively the area on $\mathcal{A}$, so that the application completion time is minimized. Since the algorithm in [25] is an expensive solution to be used inside a metaheuristic search, we propose in this paper an algorithm to

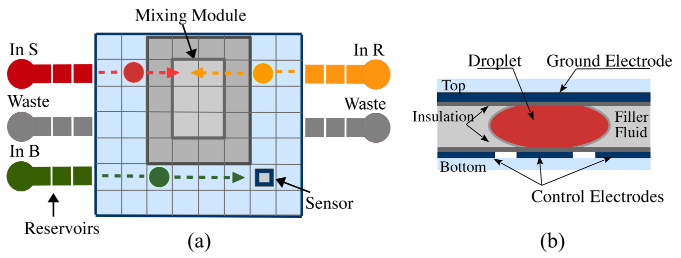


Fig. 1. Biochip architecture model. (a) General purpose biochip. (b) Droplet movement.

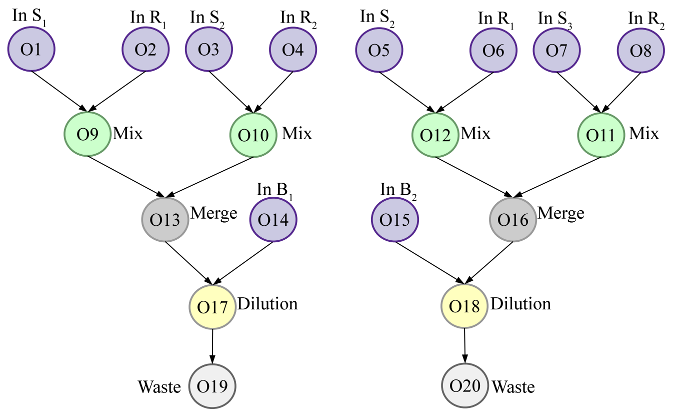


Fig. 2. Biochip application model.

incrementally build a library starting from an existing library $\mathcal{L}'$ determined for the previously visited architecture $\mathcal{A}'$, and incrementally updating $\mathcal{L}'$ for the current architecture $\mathcal{A}$.

The biochip architecture and application models are presented in the next section. The architecture synthesis problem is presented in Section III using a motivational example. Our proposed heuristic approach based on TS is presented in Section VI and in Section IV we discuss the criteria used to evaluate each architecture solution visited by TS. Section V presents our proposed placement algorithm for CRMs. We evaluate the proposed architecture synthesis in Section VII and in Section VIII we present our conclusions.

II. System Model

A. Biochip Architecture

In a DMB, a droplet is sandwiched between a top ground-electrode and a bottom control-electrode as shown in Fig. 1(b). The droplets are manipulated using the EWOD principle [3]. For example, in Fig. 1(b), if the left control-electrode is activated by applying voltage, and the other control-electrodes are turned off, then the droplet will move to the left. Considering the 8×7 biochip in Fig. 1(a), a droplet can only move up, down, left, or right with EWOD, and cannot move diagonally. A biochip is typically connected to a computer (or microcontroller) and it is controlled based on an electrode actuation sequence that specifies for each time step which electrodes have to be turned on and off, in order to run a biochemical application.

In this paper, we distinguish between the physical components of a biochip, such as electrodes, reservoirs, and detectors, and the virtual devices, such as mixing and diluting modules, which are placed on the physical array of electrodes and interconnected using virtual routes. There are two types of operations: nonreconfigurable (dispensing and detection), which are bound to a specific component such as a reservoir, a detector or a sensor and reconfigurable (mixing, split, dilution, merge, and transport), which can be executed on any electrode on the biochip. The biochip from Fig. 1(a) has two waste-reservoirs and three dispensing reservoirs, one for buffer, one for sample, and one for reagent.

In this paper, we assume that the locations of the reservoirs are on the boundaries of the array of electrodes. To dispense a droplet from the reservoir, several electrodes are activated to form a "finger" droplet, which is afterward split to obtain the final droplet [26]. Sensors can be used to determine the result of the bioassay or for error detection [27], [28].

The reconfigurable operations are executed using the electrodes in the architecture array. A mixing operation is executed when two droplets are moved to the same location and then transported together according to a specific pattern. A split operation is executed instantly by keeping the electrode on which the droplet is resting turned off, while applying concurrently the same voltage on two opposite neighboring electrodes. Dilution is a mixing operation followed by a split operation.

B. Circular-Route Module

In the past, researchers have grouped the electrodes to form "virtual devices" on which the operations execute [12]. The straightforward approach used by most researchers is to consider a rectangular area of electrodes, called a "module." For example, the droplets from Fig. 1(a) are mixing on a 2×3 module. In case two droplets are on neighboring electrodes, they merge instantly. To avoid accidental merging, researchers have assumed that each module is surrounded by a "segregation border" of one-electrode thickness [see Fig. 1(a)]. All the electrodes forming such a rectangular module are considered occupied during the operation execution, and cannot be used by other operations. However, an operation can execute anywhere on the array, and it is not necessarily confined to a rectangular area, as is the case with the routing-based compilation approach [24], which allows operations to execute on any route.

Although the approach used for operation execution on virtual devices is orthogonal to our architecture synthesis methodology, in this paper we consider a routing-based approach [24] for the operation execution. The advantage of routing-based operation execution is that it utilizes better the available biochip area. The disadvantage of routing-based operation execution is that it makes it difficult to avoid contamination. Therefore, Maftei *et al.* [24] later advocated a routing-based operation execution constrained to a given area [29].

Similar to [29], we assume that we know the position of the droplets during the execution, i.e., the operation execution is "droplet-aware." We have decided to use the droplet-aware approach [29] because of its improved reconfigurability in case of permanent faults: the droplets are instructed to avoid the faulty electrodes. Hence, our approach is to constrain the routing-based operation execution to a given "circular route." We define a CRM as a route of one-electrode thickness which starts and ends in the same electrode, and does not intersect itself. Given a CRM, a droplet will move repeatedly on the

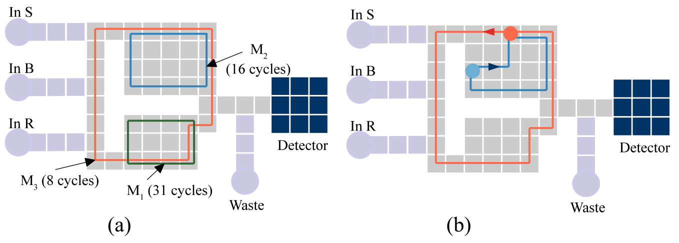


Fig. 3. Example of circular-route modules. (a) Application-specific architecture. (b) Adjusted route to avoid droplet merging.

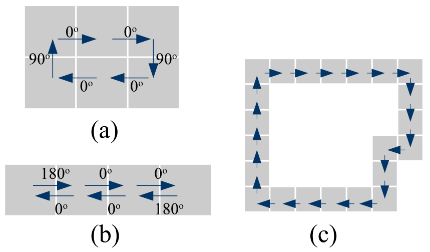


Fig. 4. Module decomposition approach. (a) 2×3 module, $t = 6.1$ s. (b) 1×4 module, $t = 4.6$ s. (c) Circlular-route module, $t = 2.2$ s.

route until the operation has completed. We denote such a CRM with M_i. Fig. 3(a) shows three examples of CRMs, M_1, M_2, and M_3.

In a CRM, only the electrode holding the droplet and the adjacent electrodes are considered occupied (to avoid accidental merging). The rest of the electrodes assigned to such a CRM are not considered occupied, and can be used for other operations. As a consequence, the routes for different operations may overlap over several electrodes. To avoid droplet merging at the intersection points, we instruct one of the droplets to take a detour as shown in Fig. 3(b) or to wait until the other droplet passed by.

We use the module decomposition approach proposed in [24] to estimate the operation execution time for each CRM. In [24], the droplet movement during an operation is decomposed into basic movements and the impact of each basic movement on the operation execution is calculated. As seen in Fig. 4(a), on a 2×3 mixer a cycle is completed by four forward movements (0°) and two turns (90°). Using an experimentally determined library that contains information about the execution times of the operations, the method proposed in [24] estimates, for each movement, the percentage completion toward operation execution.

Thus, we can determine p^0_{cycle}—the percentage toward operation execution for a cycle when there are no faults, using the following equation:

$$p^0_{\text{cycle}} = n^1_{0^\circ} \times p^1_{0^\circ} + n^2_{0^\circ} \times p^2_{0^\circ} + n_{180^\circ} \times p_{180^\circ} + n_{90^\circ} \times p_{90^\circ} \tag{1}$$

where $p^1_{0^\circ}$, $p^2_{0^\circ}$, p_{180°, and p_{90° are the percentages toward operation completion for a forward movement for one electrode, a forward movement for at least two consecutive electrodes, a backward movement and a turn, respectively, and $n^1_{0^\circ}$, $n^2_{0^\circ}$, n_{180°, and n_{90° are the number of forward movements for one electrode, forward movements for at least two consecutive electrodes, backward movements and turns, respectively. Then we determine n_i—the minimum number of times the droplets have to rotate on a given circular route to achieve at least 100% operation completion. Fig. 3(a) shows n_i for each of the three CRMs, 31 for M_1, 16 for M_2, and 8 for M_3. The total execution time is obtained by multiplying n_i with the time needed to complete one rotation. For example, for the route depicted in Fig. 4(c), the droplets need to cycle ten times in order to complete the mixing operation, resulting in an execution time of $t = 2.2$ s. We have used the following values for the percentages toward operation completion: $p^1_{0^\circ} = 0.29\%$, $p^2_{0^\circ} = 0.58\%$, $p_{180^\circ} = -0.5\%$, and $p_{90^\circ} = 0.1\%$ [24].

In order to determine the application completion time, our compilation uses a module library $\mathcal{L}$, which provides the shape of each CRM M_i and the corresponding execution time needed for each operation. For example, as seen in Table III, which is the CRM library for the architecture from Fig. 7(b), the execution time for a mixing operation on M_1 is 2.7 s if no faults are present. Columns 4 and 5 in Table III present the estimated execution times of the operations for $k = 1$ and 2 faults, respectively. All past work in compilation of DMBs considers the library given as input. The only exception is [30], where additional sensing systems are used to detect online when an operation has finished executing. In Section V, we propose an algorithm to determine for a given application-specific architecture a CRM library that estimates the operation execution time in case of k permanent faults.

C. Biochemical Application Model

A biochemical application is modeled using a directed acyclic graph $\mathcal{G}$, where the nodes represent the operations, and the edges represent the dependencies between them. Let us consider the application graph from Fig. 2, which has 20 operations. The directed edge between O_{17} and O_{19} signifies that operation O_{17} has to finish before operation O_{19} starts executing. Operation O_{19} uses the output droplet issued by operation O_{17}. The input operations dispense the droplets from the corresponding reservoirs. For instance, operation O_1 from Fig. 2, dispenses a droplet from the S_1 reservoir on the biochip illustrated in Fig. 5.

III. Problem Formulation

A. Input Libraries

In order to determine if the application satisfies its timing constraints even in the case of k faults, we determine a CRM library (see Section V) that estimates the operation execution time for maximum k faults. As mentioned, we use the module decomposition approach proposed by Maftei *et al.* [24], which uses the application-specific parameters $p^1_{0^\circ}$, $p^2_{0^\circ}$, p_{180°, and p_{90°, as explained in Section II-B. We consider that these parameters are given by the designer in a parametric library $\mathcal{P}$.

The objective of an architecture synthesis is to determine a minimum cost architecture. Our solution in [10] used a simplified cost model that considered only the cost of the physical components. In this paper, we propose an improved cost model that also considers the fluidic cost.

When an application is executed, all dispensing reservoirs are fully loaded, thus the fluidic cost depends on the number and the capacity of the dispensing reservoirs. If we use a larger number of reservoirs, we can increase the parallelism and thus complete the application faster because the dispensing operations execute slower that mixing/dilution operations

TABLE I
EXAMPLE OF COMPONENT LIBRARY $\mathcal{M}$ [31]

Name	Cost	Area (mm^2)	Time (s)
Electrode	1	1.5×1.5	N/A
Reservoir$_1$ (1 μL)	6.66	5×3	2
Reservoir$_2$ (10 μL)	16.6	7.5×5	2
Reservoir$_3$ (50 μL)	33.3	7.5×10	2
Reservoir$_4$ (100 μL)	52	15×7.5	2
Capacitive Sensor	1	1.5×4.5	0
Optical Detector	9	4.5×4.5	30

TABLE II
FLUIDIC LIBRARY $\mathcal{F}$ [32]

Fluid Name	Cost/μL
Sample (DNA$_1$)	2.47
Sample (DNA$_2$)	3.3
Reagents*	0.6

(e.g., for the colorimetric protein assay dispensing executes in 7 s, while mixing executes in 3 s on a 2×3 mixer) [33]. The cost of reagents is generally expensive and can reach up to 70% of the biochip cost [32], [34]. In addition, hard-to-obtain samples (e.g., from newborn babies, endangered species, and unique specimens), also have high cost. Hence, it is important to minimize the use of samples and reagents in order to reduce the total cost of a biochip architecture.

To determine the cost of an architecture, our model uses a component library $\mathcal{M}$ and a fluidic library $\mathcal{F}$, provided by the designer. The library $\mathcal{M}$ contains a list of the physical components available to design a biochip. An example component library is Table I, where, for each physical component mentioned in column 1, column 2 presents the costs expressed in the unit cost of an electrode, column 3 presents the dimensions and column 4 presents the execution time. The electrode component (row 1 in Table I) can be reconfigured to perform various operations, thus the electrode has a "N/A" execution time. The operations that can be performed on the electrode components and their execution times are specified in the module library $\mathcal{L}$ (see Table III).

A fluidic library $\mathcal{F}$ contains for each input fluid the cost per μL expressed in the unit cost of an electrode. After the binding of the operations is decided, we use library $\mathcal{M}$ to calculate the total cost for each input fluid. For example, Table II presents the fluidic library for the polymerase chain reaction (PCR) assay. Let us assume that the sample fluid DNA$_1$ with a cost of 2.47 units/μL (row 1 in Table II) is bound to Reservoir$_2$, which has a capacity of 10 μL (row 3 in Table I). The total cost for DNA$_1$ is 24.7 units.

Section IV presents how the libraries $\mathcal{M}$ and $\mathcal{F}$ are used by our proposed cost model for a biochip architecture.

B. Problem Formulation

In this paper, we address the following problem. Given as input a biochemical application $\mathcal{G}$ with a deadline $D_{\mathcal{G}}$, the parametric library $\mathcal{P}$, the component library $\mathcal{M}$, the fluidic library $\mathcal{F}$, and maximum k permanent faults to be tolerated, we are interested to synthesize a fault-tolerant physical architecture $\mathcal{A}$, such that the cost of $\mathcal{A}$ is minimized and the application completion time $\delta_{\mathcal{G}}^{k}$ is within the deadline $D_{\mathcal{G}}$ for any occurrence of the k faults.

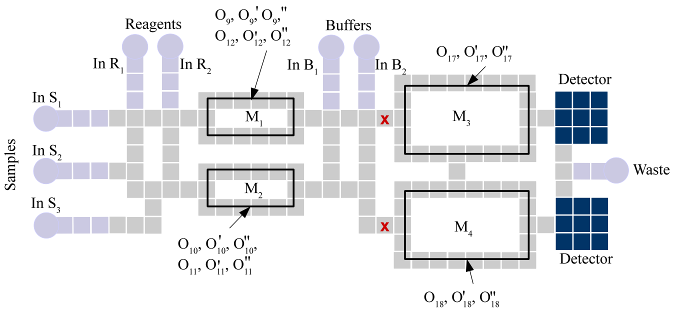

Reagents
O_9, O_9', O_9''
O_{12}, O'_{12}, O''_{12}
Buffers
In R_1
In R_2
In B_1
In B_2
O_{17}, O'_{17}, O''_{17}
Detector
In S_1
M_1
M_3
Samples
In S_2
Waste
M_2
In S_3
M_4
O_{10}, O'_{10}, O''_{10},
O_{11}, O'_{11}, O''_{11}
Detector
O_{18}, O'_{18}, O''_{18}

Fig. 5. Application-specific biochip architecture.

C. Motivational Example

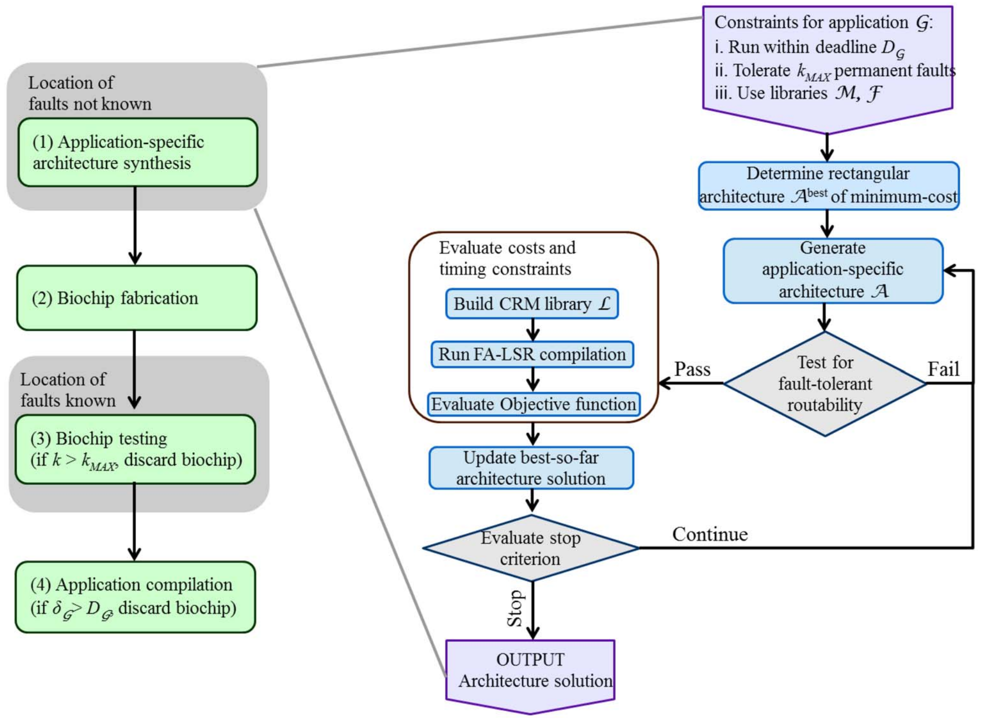
Let us consider an application graph $\mathcal{G}$ obtained by repeating three times the graph from Fig. 2. The entire graph $\mathcal{G}$ is not depicted due to space reasons. We are interested to synthesize a physical architecture for this application, considering $k = 1$ permanent faults, such that the cost is minimized and a deadline of $D_{\mathcal{G}} = 22$ s is satisfied.

So far, researchers have considered only general-purpose biochips of rectangular shape. To complete $\mathcal{G}$ within deadline $D_{\mathcal{G}}$ using a rectangular architecture $\mathcal{A}_2$, we need an array of 9×16 electrodes and eight reservoirs: two for the reagent, two for the buffer, three for the sample, and one for the waste. The rectangular architecture $\mathcal{A}_2$ has 168 electrodes. We used the module library in [10] and obtained an execution time for $\mathcal{G}$ on $\mathcal{A}_2$ of 18.78 s, which is satisfying the deadline.

However, the number of electrodes can be reduced if we use an application-specific architecture $\mathcal{A}_1$, such as the one in Fig. 5, of only 128 electrodes, reducing with 23.8% the number of electrodes of $\mathcal{A}_2$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same number of reservoirs, i.e., both architectures have identical fluidic cost, we compare $\mathcal{A}_1$ and $\mathcal{A}_2$ only in terms of number of electrodes. For architecture $\mathcal{A}_1$, we determined manually the following worst-case completion times in case of $k = 1$ permanent faults: 2.59 s for a mix operation on M_1 and M_2, 5.16 s for a dilution operation on M_1 and M_2, 2.4 s for a mix operation on M_3 and M_4, and 4.47 s for a dilution operation on M_3 and M_4. The binding of operations is shown in the figure; we replicate three times the graph in Fig. 2, hence, for every O_i, we have O_i' and O_i''. The completion time of $\mathcal{G}$ on architecture $\mathcal{A}_1$ is $\delta_{\mathcal{G}} = 18.87$ s, within the deadline $D_{\mathcal{G}} = 22$ s. In addition, $\mathcal{A}_1$ is also fault-tolerant to $k = 1$ permanent faults, i.e., in the worst-case fault scenario, when the fault is placed such that it leads to the largest delay on $\delta_{\mathcal{G}}$, the application completes in $\delta_{\mathcal{G}}^{k=1} = 20.01$ s, which satisfies the deadline.

We assume that our architecture synthesis is part of a methodology, outlined in Fig. 6, which has the following steps.

1) We synthesize an application-specific architecture $\mathcal{A}$ for an application $\mathcal{G}$ with a deadline $D_{\mathcal{G}}$, considering a maximum number of permanent faults k that have to be tolerated. Since the architecture synthesis is performed before the fabrication (step 2) and testing (step 3), the locations of permanent faults are not known.
2) We fabricate the biochips with application-specific architecture $\mathcal{A}$, obtained during the previous step.
3) All the biochips are tested to determine if they have permanent faults using testing techniques such as the ones proposed in [6]. If there are more than k faults, the



Location of faults not known
(1) Application-specific architecture synthesis
(2) Biochip fabrication
Location of faults known
(3) Biochip testing (if $k > k_{MAX}$, discard biochip)
(4) Application compilation (if $\delta_{\mathcal{G}} > D_{\mathcal{G}}$ discard biochip)
Constraints for application $\mathcal{G}$:
i. Run within deadline $D_{\mathcal{G}}$
ii. Tolerate k_{MAX} permanent faults
iii. Use libraries $\mathcal{M}$, $\mathcal{F}$
Determine rectangular architecture $\mathcal{A}^{best}$ of minimum-cost
Generate application-specific architecture $\mathcal{A}$
Test for fault-tolerant routability
Pass
Fail
Evaluate costs and timing constraints
Build CRM library $\mathcal{L}$
Run FA-LSR compilation
Evaluate Objective function
Update best-so-far architecture solution
Evaluate stop criterion
Continue
Stop
OUTPUT Architecture solution

Fig. 6. Methodology and architecture synthesis overview.

biochip is discarded. The exact locations of permanent faults are determined during this step.

4) We perform a compilation of application $\mathcal{G}$ on $\mathcal{A}$ to obtain the electrode actuation sequence. Since the locations of permanent faults are known, we can use any compilation implementation to determine the actual completion time $\delta^k_{\mathcal{G}}$. In case $\delta^k_{\mathcal{G}}$ exceeds the deadline $D_{\mathcal{G}}$, the biochip is discarded.

In this paper, our architecture synthesis is based on TS, presented in Section VI. The proposed metaheuristic explores the solution space using design transformations called moves, which are applied to the current architecture solution in order to obtain neighboring architecture alternatives. A new architecture solution is accepted if it improves the current solution. The metaheuristic continues to apply the moves on the determined current solution and use the objective function to evaluate the obtained neighboring architectures. The search terminates when a stop criterion is satisfied.

IV. Architecture Evaluation

Our TS synthesis, presented in Section VI, evaluates each architecture in terms of: 1) routability; 2) cost; and 3) timing constraints. Each of the evaluation criteria is presented in the next paragraphs.

A. Routability Evaluation

The routability evaluation guarantees that an architecture cannot become disconnected due to permanent faults, i.e., routing of droplets to the desired destination is still possible. For example, the architecture in Fig. 5 becomes disconnected when the two electrodes marked with a red "x" are affected by faults. If an architecture can be disconnected by k faults, it should be discarded and in this case the evaluation of the other criteria (cost and timing constraints) is no longer meaningful.

We say that an architecture passes the routability test, if, in any scenario of k permanent faults, there is at least one route that connects each nonfaulty electrode to the other nonfaulty electrodes. Our implementation adapts the polynomial time $O(kn^3)$ algorithm from [35], that tests the k-vertex connectivity of a graph. We model the architecture as a graph, in which the nodes represent the electrodes and the edges represent the direct connection between them. An electrode is considered connected only to its top, bottom, left, and right neighbors, and not diagonally, since a droplet cannot be moved diagonally with EWOD. The algorithm from [35] tests if the graph remains connected in case of removal of k nodes. For example, the architecture in Fig. 5 is still connected for $k = 1$, but becomes disconnected for $k = 2$, e.g., if the two faults happen as indicated with the red x.

B. Cost Evaluation

We evaluate the cost of an architecture using the following equation:

$$\text{Cost}_{\mathcal{A}} = \sum N_{M_i} \times \text{Cost}_{M_i} + \sum N_{R_i} \times \text{Cost}_{R_i} \qquad (2)$$

where N_{M_i} is the number of physical components of type M_i, Cost_{M_i} is the cost of M_i, N_{R_i} is the number of reservoirs of type R_i, and Cost_{R_i} is the cost of the input fluid for R_i.

The first term of (2) calculates the cost of the physical components and the second term calculates the cost of the input fluids. The physical components (e.g., electrodes, reservoirs, and detectors) and their unit cost are provided by the designer in a library $\mathcal{M}$ (see Table I for an example). The unit cost of the input fluids, used by the biochemical application, are specified in a fluidic library $\mathcal{F}$, such as the one in Table II.

The assumption is that all the reservoirs integrated in the cartridge are fully loaded. We ignore the cost of the controller platform because its cost is amortized over time.

C. Completion Time Evaluation

We evaluate the architecture against its timing constraints by evaluating the application completion time in case of faults ($\delta_{\mathcal{G}}^{k}$). The value of $\delta_{\mathcal{G}}^{k}$ is used in the objective function (4) as explained in Section VI. In this section, we present in detail how we determine $\delta_{\mathcal{G}}^{k}$. The value of $\delta_{\mathcal{G}}^{k}$ is obtained through compilation, a task that consists of: 1) placement, which decides the positions of CRMs on the architecture $\mathcal{A}$; 2) scheduling, which decides the order of the operations; and 3) routing, which determines the droplet routes. Our proposed compilation, called fault-aware list scheduling and routing (FA-LSR) takes as input the architecture under evaluation $\mathcal{A}$, the application $\mathcal{G}$, the CRM library $\mathcal{L}$, and the number of permanent faults k to be tolerated, and outputs the estimated completion time $\delta_{\mathcal{G}}^{k}$.

The application completion time $\delta_{\mathcal{G}}^{k}$ depends on the location of the k permanent faults. At this point in time, we do not know the position of the k faults (they will be known after fabrication and testing), so our evaluation has to estimate $\delta_{\mathcal{G}}^{k}$. This problem of finding the worst-case schedule length has been addressed in the context of transient faults on distributed multiprocessor systems, and researchers have used "fault-tolerant process graphs" to model all possible fault-scenarios [36]. Such a modeling of all possible fault-scenarios is not feasible in our case because of the interplay between the faulty-electrodes and the allocation, binding, scheduling, and placement of operations that can be affected by these faults.

Our solution in [10] determined the worst-case schedule length in a pessimistic, but safe way, by assuming that each operation is affected by k permanent faults. However, because [10] assumes the worst-case pattern of faults for all evaluated architectures, it may also eliminate potentially good low-cost architectures which, after fabrication, when the pattern of faults is known, would have proven able to run the application within its deadline. Instead of considering the worst-case scenario as in [10], FA-LSR uses an estimate for the operation execution, calculated as discussed in Section V-A. The estimation of $\delta_{\mathcal{G}}^{k}$ is not safe, i.e., it may return smaller values than the worst-case. As a consequence, our synthesis may obtain an architecture solution that for a certain pattern of faults, will not complete the application within its deadline. The actual application completion time is determined after fabrication and testing (step 4 in the methodology depicted in Fig. 6), when the pattern of faults is known. In case the application is not completed within the deadline, the biochip is discarded.

With FA-LSR, we also introduce an extension to the placement of operations (Algorithm 2 in Section V) considering circular route modules, which do not have to be rectangular, as explained in Section II-B.

Algorithm 1 presents FA-LSR. Every node from $\mathcal{G}$ is assigned a specific priority according to the critical path priority function (line 1) [37]. List contains all operations that are ready to run, sorted by priority (line 3). An operation is ready to be executed when all input droplets have been produced, i.e., all predecessor operations from graph $\mathcal{G}$ finished executing. The intermediate droplets that have to wait for the other operations to finish, are stored on the biochip. The algorithm takes each ready operation O_i and iterates through the library $\mathcal{L}$, to find the CRM M_i that can be placed at the earliest time and executes the operation the fastest (line 6). After O_i is bound to M_i (line 7), CalculateRoute (line 8) determines the route that brings the necessary droplets to M_i and O_i is scheduled (line 10). Since the droplets can pass through CRMs when routing (we use a droplet-aware approach), we need to determine only the routing time and not the actual routes. For that purpose, CalculateRoute adapts the Hadlock's algorithm [38] to determine the route lengths. List is updated with the operations that have become ready to execute (line 11). The repeat loop terminates when List is empty (line 12).

Algorithm 1: FA-LSR

```
Input: architecture $\mathcal{A}$, application graph $\mathcal{G}$, CRM library
       $\mathcal{L}$, number of permanent faults $k$
Output: application completion time $\delta_{\mathcal{G}}^{k}$
1  CriticalPathPriority($\mathcal{G}$);
2  $t = 0$;
3  $List$ = GetReadyOperations($\mathcal{G}$);
4  repeat
5      $O_i$ = RemoveOperation($List$);
6      $M_i$ = FindCRM($\mathcal{L}$);
7      Bind($O_i$, $M_i$);
8      $route_i$ = CalculateRoute($O_i$, $M_i$, $\mathcal{G}$);
9      $t$ = earliest time when $M_i$ can be placed;
10     $\mathcal{S}$ = Schedule($O_i$, $t$, $route_i$, $M_i$, $\mathcal{L}$);
11     UpdateReadyList($\mathcal{G}$, $t$, $List$);
12 until $List = \emptyset$;
13 $L_{CP}$ = CriticalExecutionPath($\mathcal{S}$);
14 $FaultTable$ = DistributeFaults($L_{CP}$, $k$, $\mathcal{S}$);
15 for $O_i$ in $FaultTable$ do
16     $k_i$ = number of faults for $O_i$;
17     UpdateSchedule($\mathcal{S}$, $O_i$, $k_i$, $\mathcal{L}$);
18 end
19 $\delta_{\mathcal{G}}^{k}$ = the finishing time of last operation in $\mathcal{S}$
```

Next, we need to decide on the fault scenarios that will give us a realistic estimate of the application completion in case of maximum k faults. As mentioned, when the synthesis is performed, the location of the permanent faults is not known. Consequently, we do not know which operations are affected by faults and what is the worst-case fault scenario. Alistar *et al.* [10] have proposed a pessimistic approach, by considering that every operation in the application suffers from k faults. Because the length of schedule $\mathcal{S}$ is given by the critical path, which is the path in graph $\mathcal{G}$ with the longest execution time, the approach we propose in this paper is to consider that the faults affect the operations that are on the critical path—scenario that will impact most the application completion time.

First, we determine L_{CP}—the list of operations on the critical path. The k faults are distributed among the operations in L_{CP} by the DistributeFaults function such that the impact of the faults is maximized. DistributeFaults uses a greedy randomized approach [39] that takes each of the

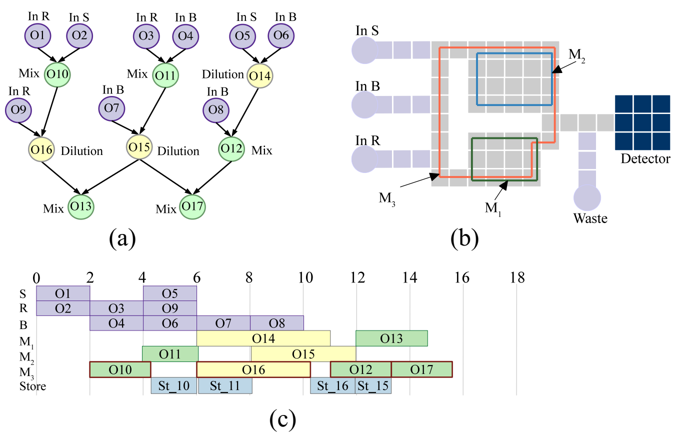


Fig. 7. Compilation example. (a) Application. (b) Placement of CRMs. (c) Schedule for $k = 1$.

k faults and after evaluating each operation in L_{CP}, distributes the fault to the operation that delays the most the application completion time. Depending on the criticality of specific operations, it may be the case that an operation is affected by more than one fault. Furthermore, if we distribute a fault to operation $O_i \in L_{\text{CP}}$, i.e., O_i executes on a faulty CRM M_i, then all operations executing on CRMs that intersect M_i will also be considered affected by a fault. The faulty operations and their corresponding number of faults are stored in FaultTable. Finally, for each operation $O_i \in$ FaultTable affected by k_i faults, the schedule is updated (line 17) with the corresponding estimated execution time stored in the library $\mathcal{L}$, which is determined by the algorithm presented in Section V-A. The application completion time $\delta_{\mathcal{G}}^k$ is the finishing time of the last operation in the schedule table (line 19).

Let us assume that we have to compile the application $\mathcal{A}$ from Fig. 7(a) on the architecture from Fig. 7(b) considering $k = 1$ permanent faults. We use the algorithm presented in Section V to determine for $\mathcal{A}$ the CRM library $\mathcal{L}$ shown in Table III, which contains the placement of CRMs, the execution time for $k = 0$ (no faults) and the estimated execution time for $k = 1$ and $k = 2$. For simplicity reasons, in this example we ignore routing and consider that there are no contamination constraints. In order to avoid congestion, the dispensing operations are scheduled only when the corresponding dispensed droplets are needed.

At time $t = 2$ s mixing operation O_{10} has the highest priority among all the ready operations (an operation is ready if all its input droplets have arrived). For O_{10}, the CRM M_3 [see Fig. 7(b)] is selected from library $\mathcal{L}$ (Table III), since it finishes the mixing operation the fastest. At time $t = 4.08$ s, operation O_{10} finishes executing. However, the successor of O_{10}, operation O_{16}, is not ready to execute because the other predecessor operation O_{09} has not finished executing. At $t = 6$ s, O_{09} finishes executing, and List is updated with operation O_{16}, which becomes ready to execute.

First, FA-LSR will produce a schedule of 15.16 s (lines 4–12 in Algorithm 1). The schedule of operations is presented as a Gantt chart, where the start time of an operation is captured by the left edge of the respective rectangle, and the length of the rectangle represents the duration. Next, DistributeFaults will distribute the $k = 1$ faults to operation O_{17}, since it results in

TABLE III
CRM LIBRARY $\mathcal{L}$ FOR THE ARCHITECTURE FROM FIG. 7(b)

Operation	CRM	Time (s) k=0	Time (s) k=1	Time (s) k=2
Mix	M_1	2.7	4	15.81
	M_2	2.1	2.4	3
	M_3	2.08	2.3	2.64
Dilution	M_1	5	6.8	16.68
	M_2	3.92	4.44	5.25
	M_3	3.9	4.14	4.4

the greatest increase in schedule length. Consequently, operations O_{10}, O_{16}, and O_{12}, which execute on the same CRM as O_{17}, suffer from $k = 1$ permanent faults. The schedule length is updated with the execution times for the faulty operations $C_i^{k=1}$, taken from library $\mathcal{L}$ (column 4 in Table III). As shown in the schedule from Fig. 7(c), the completion time $\delta_{\mathcal{G}}^1$ is 15.6 s.

V. Building Library of Circular-Route Modules

In this section, we present our approach to determine the CRM library $\mathcal{L}$, which is used by the FA-LSR compilation (Algorithm 1 in Section IV-C) to obtain the application completion time $\delta_{\mathcal{G}}^k$, and thus, check if the timing constraints are satisfied. For each determined CRM, its shape and the corresponding placement on the biochip are also stored. Hence, the placement task does not need to be implemented during the compilation step. For example, the CRM M_3 determined for the architecture in Fig. 7(b), has the shape of the following dimensions: $8 \times 7 \times 6 \times 2 \times 3 \times 6$, with the corners placed at coordinates: (0, 0), (0, 7), (6, 7), (6, 2), (5, 2), and (5, 0).

In [25], we have proposed an algorithm that builds a library $\mathcal{L}$ of CRMs for the application-specific architecture $\mathcal{A}$, which does not take faults into account. Since the algorithm in [25] is time consuming and hence cannot be used inside a metaheuristic search, we propose in this paper an incremental library build (ILB) algorithm that starts from an existing library $\mathcal{L}'$ determined for the previously visited architecture solution $\mathcal{A}'$, and incrementally updates $\mathcal{L}'$ to obtain $\mathcal{L}$ for the current architecture $\mathcal{A}$. This is possible because during the TS-based design space exploration, a new architecture $\mathcal{A}$ is generated by applying gradual transformations to the previous solution $\mathcal{A}'$. Hence, the newly generated architecture $\mathcal{A}$ shares a similar layout with $\mathcal{A}'$. Consequently, the corresponding CRM library $\mathcal{L}$ can be built by incrementally updating $\mathcal{L}'$, that is the library previously determined for $\mathcal{A}'$.

A CRM is defined as a circular-route of electrodes (see Section II-B), and we denote a CRM M as a set of distinctive electrodes $\{e_1, e_2, \ldots, e_n\}$. An electrode e_n is a neighbor-electrode to e_m, if e_m can be reached directly from e_n. Note that a droplet cannot move diagonally. We define a chain of electrodes $\mathcal{R}$ as a set of consecutive neighboring electrodes that are all situated on the same coordinate axis (vertical or horizontal). For example, in the case of the application-specific biochip depicted in Fig. 8(a), the electrodes marked with x form a chain, while the ones marked with "y" do not form a chain. Note that a chain can also consist of a single electrode.

Our proposed ILB (Algorithm 2) is general, i.e., it can be used for any transformation involving a set of electrodes $\mathcal{E}$, which is decomposed in chains of electrodes (line 1). Each chain

Algorithm 2: ILB

```
Input: architecture $\mathcal{A}$, library $\mathcal{L}'$, set of electrodes $\mathcal{E}$, $k$
       faults
Output: new CRM library $\mathcal{L}$
1  $L_C$ = DecomposeInChains($\mathcal{E}$, $\mathcal{A}$);
2  for $\mathcal{R}_j \in L_C$ do
3      $L_M$ = DetermineCRMs($\mathcal{L}'$, $\mathcal{R}_i$);
4      for $M_i \in L_M$ do
5          if $\mathcal{R}_j$ is added then
6              $\mathcal{H}_i$ = FindNeighborChain($M_i$, $\mathcal{R}_j$);
7              AdjustCRM($M_i$, $\mathcal{H}_i$);
8          end
9          if $\mathcal{R}_j$ is removed then
10             route = DetermineRoute($M_i$, $\mathcal{A}$);
11             ReconstructCRM($M_i$, route);
12         end
13         EstimateOpExecution($M_i$, $k$);
14         UpdateLibrary($\mathcal{L}$, $M_i$);
15     end
16 end
```

of electrodes $\mathcal{R}_j \in \mathcal{E}$, can be in one of the two cases: 1) $\mathcal{R}_j$ is added to $\mathcal{A}'$ or 2) $\mathcal{R}_j$ is removed from $\mathcal{A}'$. For example, the transformation applied on architecture $\mathcal{A}'$ [Fig. 10(a)] to obtain the architecture $\mathcal{A}$ [Fig. 10(b)] can be decomposed into the following: adding the chains of electrodes $\mathcal{R}_1$ and $\mathcal{R}_4$ and removing the chains of electrodes $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_5$.

In both cases, ILB determines the CRMs from $\mathcal{L}'$ on which $\mathcal{R}_j$ has an impact (line 3) and then we adjust those CRMs (lines 4–12) so that the adjustment improves the operation execution time. Next, for each newly adjusted CRM, ILB estimates the operation execution time in case of maximum k permanent faults (line 13) and updates the library (line 14).

Let us present in detail how our proposed algorithm works. First, ILB decomposes the set of electrodes $\mathcal{E}$ in chains of electrodes which are stored in the list L_C. Next, for each chain of electrodes $R_j \in \mathcal{E}$, DetermineCRM (line 3) determines L_M—the list of CRMs on which $\mathcal{R}_j$ has an impact. The strategy used by the DetermineCRM function depends on whether $\mathcal{R}_j$ is added or removed. For the first case, when $\mathcal{R}_j$ is added to $\mathcal{A}'$, DetermineCRMs selects from the library $\mathcal{L}'$, the CRMs impacted by this move, i.e., the CRMs that include at least one neighbor-electrode to an electrode in $\mathcal{R}_j$. Those CRMs are stored in the list L_M. In Fig. 8(b), the CRM M_1 is neighboring three electrodes from $\mathcal{R}_j$, and consequently, M_1 is added to L_M. In case (2), when $\mathcal{R}_j$ is removed from $\mathcal{A}'$, DetermineCRMs (line 3) adds to L_M the CRMs that contain any electrode in $\mathcal{R}$. In Fig. 9(a), the CRM M_2 contains electrodes in $\mathcal{R}_j$ (the three hashed electrodes), and consequently M_2 is included in L_M.

Next, ILB adjusts each $M_i \in L_M$ so that the operations will complete faster. Reconfigurable operations (e.g., mixing and dilution) complete faster when the forward movement of the droplets is prioritized and the backward movement is avoided [40]. Hence, for case (1), the newly added electrodes $\in \mathcal{R}_j$ are used to adjust the CRMs so that forward movements are prioritized. To do that, FindNeighborChain (line 6) inspects all electrodes in $\mathcal{R}_j$ to determine $\mathcal{H}_i$—the longest chain of electrodes that has both ends as neighbor-electrodes

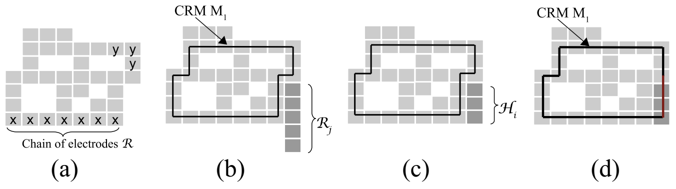


Fig. 8. Adjusting a CRM in case (1). (a) Previous architecture. (b) Chain of electrodes $\mathcal{R}_j$. (c) Chain of electrodes $\mathcal{H}_i$. (d) Adjusted CRM.

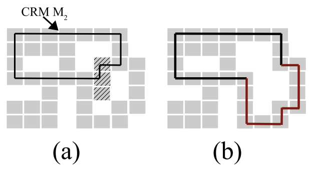


Fig. 9. Reconstructing a CRM in case (2). (a) Chain of electrodes $\mathcal{R}_j$. (b) Reconstructed CRM.

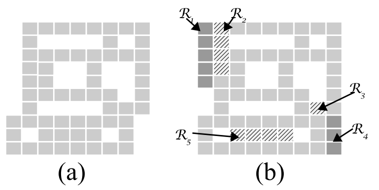


Fig. 10. Example of complex transformation. (a) Previous architecture $\mathcal{A}'$. (b) Current architecture $\mathcal{A}'$.

to an electrode in $\mathcal{M}_j$. AdjustCRM includes $\mathcal{H}_i$ in M_j only if the adjustment results in a greater count of forward movements. For the example in Fig. 8(b), we have determined the chain of electrodes $\mathcal{H}_i$ in Fig. 8(c), and the adjusted M_1 in Fig. 8(d).

In case (2), when $\mathcal{R}_j$ is removed from $\mathcal{A}'$, the CRMs $M_i \in L_M$ have to be rerouted to avoid the removed electrodes. After the removal of the hashed electrodes in Fig. 9(a), the route of M_2 has to be rerouted as shown in Fig. 9(b). Since ILB is used inside a search metaheuristic, we are more interested in finding a new route fast, then in finding the shortest route. Hence, in order to find a new connecting route for M_i, DetermineRoute (line 10) uses Soukup's algorithm [38].

In case such a route cannot be found, M_i is removed from L_M, otherwise ReconstructCRM (line 11) includes the route in M_i. Next, for each CRM $\in L_M$, EstimateOpExecution (line 13) determines a parametric estimation of the operation execution time in case of maximum k permanent faults. The library is updated (line 14) and it can be used by the FA-LSR compilation to determine the application completion time.

The next section presents in detail the algorithm used by EstimateOpExecution.

A. Estimation of Operation Execution Time in Case of Faults

When building the library of CRMs $\mathcal{L}$ for the current application-specific architecture, we need to determine for each CRM $M_i \in \mathcal{L}$ the corresponding operation execution time C_i^f, which is the time needed by a reconfigurable operation (e.g., mix and dilution) to complete on M_i considering f permanent faults, $f = 1$ to k, where k is the maximum number of permanent faults. The value of C_i^f depends on the pattern of permanent faults.

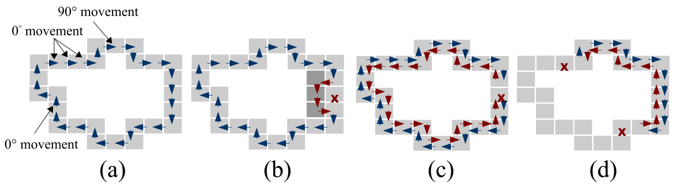


Fig. 11. Estimation of operation execution time in case of k faults. (a) $t = 3.9$ s (no faults). (b) $t = 5.04$ s ($k = 1$). (c) $t = 4.8$ s ($k = 1$). (d) $t = 5.28$ s ($k = 2$).

When our architecture synthesis is run, the location of the faults is not known so we need a method to estimate C_i^f. To solve this problem, Alistar *et al.* [25] used exhaustive search to determine the pattern of permanent faults which maximizes C_i^k. In this paper, we propose a faster method to determine an estimate of C_i^f, which is less pessimistic than the worst-case value. Consequently, it might happen that, if a specific pattern of faults has occurred during fabrication, our estimated C_i^f is less than the actual execution time. We can determine if that is the case by running a post-fabrication compilation and discarding the biochip if the timing constraints are not satisfied.

Hence, the function EstimateOpExecution (line 13) called by ILB (Algorithm 2), determines for a CRM M_i a parametric estimation of the operation execution time C_i^f, where $f = 1$ to k, assuming M_i is placed over an area with f faulty electrodes.

Let us denote with p^0_{cycle} the percentage toward operation completion for a cycle when there are no faults. The value of p^0_{cycle} is obtained using (1) as explained in Section II-B. We need to estimate p^f_{cycle}—the percentage toward operation completion for a cycle when there are f permanent faults. Once we know p^f_{cycle}, we calculate n_i^f—the number of cycles needed to achieve 100% operation execution. The value of C_i^f is obtained by multiplying n_i^f with the time needed for one cycle p^f_{cycle}.

In case a CRM contains faulty electrodes, the droplets need to be rerouted in order to avoid the permanent faults. Let us consider that the CRM in Fig. 11(a) has the faulty electrodes marked with x in Fig. 11(b). In order to avoid the faults, the droplets are instructed to take a detour, as shown in Fig. 11(b). Since the position of the faults is not known, we have used in [10] an exhaustive search to determine the execution time for the worst-case fault pattern occurring in a CRM. As an alternative, in this paper we propose a faster estimation heuristic, which, instead of performing an exhaustive search, makes some simplifying assumptions about the impact of the worst-case fault patterns on the operation execution.

One assumption is that the permanent faults form a pattern which can only be avoided using backward movements (180° turns). Backward movements lengthen the operation execution time C_i, because they induce flow reversibility. In most of the cases, using a backward movement can lead to a larger increase in execution time than taking a detour. Since we do not know the exact location of the faults, and consequently whether detour-routes are possible, it is our assumption, as mentioned, that faults are positioned in such locations that they can be avoided only using backward movements. Hence, the droplets will be routed back and forth between two of the f faults, as shown in Fig. 11(d). Consequently, the cycle of droplets on the CRM is reduced to the distance between two of the f faults. In case $f = 1$, the droplets will be routed as shown in Fig. 11(c).

Another assumption is that, if there are more faults ($f > 1$), they are located such that they lead to the "most damage," i.e., the largest increase in C_i. We assume that this happens when the faults are located at equal distance on the CRM, as shown in Fig. 11(d). Our assumption is based on the fact that a route with a higher frequency of backward movements will need more time to complete the operation.

Considering the assumptions mentioned above, we estimate p^f_{cycle} using the following equation:

$$p^f_{\text{cycle}} = p^0_{\text{cycle}}/f - 2 \times p^2_{0^\circ} + p_{180^\circ} \tag{3}$$

where p^0_{cycle}, p_{180°, and $p^2_{0^\circ}$ are the percentages toward operation completion for a cycle with no faults, a backward movement and a forward movement for at least two consecutive electrodes, respectively. Since we consider that the f faults are located at equal distance, we obtain in the first term in (3), a rough estimation of p^f_{cycle} by dividing p^0_{cycle} to f. However, to be more precise, we take into account that two electrodes are occupied by faults [second term in (3)] and that a 180° turn is needed [third term in (3)]. For the second term in (3), we assumed we are loosing electrodes which contribute most toward operation completion (i.e., the completion percentage is $p^2_{0^\circ}$).

The values of p^0_{cycle} [determined using (1), see Section II-B], $p^2_{0^\circ}$ and p_{180° depend on the operation type and on the fluids used the operation. Hence, (3) determines for each CRM a parametric estimation of the execution time, where the parameters are the percentages $p^1_{0^\circ}$, $p^2_{0^\circ}$, p_{180°, and p_{90°. Once the binding of the operations is decided, i.e., we know which operations are assigned to each CRM, then we introduce in (3) the corresponding values for the parameters, which are provided, as mentioned, in the parametric library $\mathcal{P}$. We assume that the values for p^0_{cycle}, $p^2_{0^\circ}$, and p_{180° are given by the designer.

For our example, we use the following values: $p^1_{0^\circ} = 0.29\%$, $p^2_{0^\circ} = 0.58\%$, $p_{180^\circ} = -0.5\%$, and $p_{90^\circ} = 0.1\%$ [24]. Considering $k = 2$ permanent faults for the CRM M_1 in Fig. 11(a), we estimate, using (3), the following execution times: $C_1^1 = 4.8$ s and $C_1^2 = 5.28$ s [see Fig. 11(c) and (d)]. As presented in Section IV-C, these operation execution values C_i^f are used inside our FA-LSR compilation (Algorithm 1) to determine the application completion time.

VI. Tabu Search-Based Architecture Synthesis

We use a TS [41] metaheuristic for our application-specific architecture synthesis optimization problem, which is an NP-complete problem.

TS takes as input the application graph $\mathcal{G}$, the physical components library $\mathcal{M}$, the number of permanent faults to be tolerated k and the CRM library $\mathcal{L}$ and produces the architecture $\mathcal{A}$ that minimizes the objective function [see (4)]. TS explores the solution space using design transformations, called moves, to generate the neighborhood $\mathcal{N}$ of the current solution. To prevent cycling due to revisiting solutions, tabu moves are stored in a short-term memory of the search,

Algorithm 3: Tabu Search-Based Architecture Synthesis

```
Input: application $\mathcal{G}$, library $\mathcal{M}$, number of permanent
        faults $k$, deadline $D$ Output: architecture $\mathcal{A}^{\text{best}}$
1  $\mathcal{A}^{\text{best}}$ - rectangular architecture that minimizes the
   Objective function;
2  $\mathcal{L}^0$ = BuildCRMLibrary($\mathcal{A}^{\text{best}}$);
3  $t = 0$;
4  while $t \leq$ time limit do
5      $\mathcal{N}$ = GenerateNeighborhood($\mathcal{A}^{\text{current}}$, TabuList);
6      Update(TabuList, $\mathcal{N}$);
7      for each $\mathcal{A}^{current} \in \mathcal{N}$ do
8          $\mathcal{L}$ = ILB($\mathcal{A}^{\text{current}}$, $\mathcal{L}^0$, $k$);
9          $\delta_{\mathcal{G}}^k$ = FA-LSR($\mathcal{A}^{\text{current}}$, $\mathcal{G}$, $\mathcal{L}$, $k$);
10         if Objective($\mathcal{A}^{current}$) < Objective($\mathcal{A}^{best}$) then
11             $\mathcal{A}^{\text{best}} = \mathcal{A}^{\text{current}}$;
12         end
13         $\mathcal{L}^0 = \mathcal{L}$;
14     end
15     if diversification is needed then
16         $\mathcal{A}^{\text{current}}$ = ApplyDiversificationMove($\mathcal{A}^{\text{current}}$);
17         Restart(TabuList, $\mathcal{A}^{\text{best}}$ , $\mathcal{A}^{\text{current}}$ );
18     end
19 end
```

namely the tabu list, which has a fixed dimension, called tabu tenure. However, it may happen that most of the search is done locally, exploring only a restricted area of the search space. In that case, TS uses diversification to direct the search toward unexplored regions. Thus, a diversification move applied to the best-known solution, and the search is restarted from that point.

Algorithm 3 illustrates our TS-based architecture synthesis. We first determine the rectangular architecture of minimum cost $\mathcal{A}^{\text{best}}$ that can run the application within deadline (line 1). For that, we use exhaustive search by starting from the rectangular architecture of minimum acceptable size and incrementally increasing the dimensions. Each intermediate architecture is evaluated in terms of $\delta_{\mathcal{G}}$. The exhaustive search stops when we obtain an architecture that can run the application within the deadline.

We use the algorithm from [25] to build the initial library $\mathcal{L}^0$ (line 2) for the initial architecture solution $\mathcal{A}^{\text{best}}$, from which TS starts exploring the design space. To explore the design space, GenerateNeighborhood (line 4) generates new neighbor architecture by applying moves to the current solution $\mathcal{A}^{\text{current}}$. The moves are divided in two classes: 1) moves for electrodes and 2) moves for devices. In the next paragraph, we define the moves and we illustrate some of them considering the current solution $\mathcal{A}^{\text{current}}$ in Fig. 12(a). Note that we mark the added electrodes with a darker shade of gray and we hash the removed electrodes.

1) We define the moves for electrodes as follows.
   a) Adding a chain of electrodes at a random position.
   b) Removing a chain of electrodes at a random position.
   c) Adding a chain of electrodes at the sides of the architecture, namely at the top, bottom, right, and left.
   d) Removing a chain of electrodes at the sides of the architecture, namely at the top, bottom, right, and left.
2) We define the following moves for devices.
   a) Three moves that add dispensing reservoirs: for samples, reagents, and buffers, respectively.
   b) Three moves that remove dispensing reservoirs: for samples, reagents, and buffers, respectively.
   c) Adding a detector.
   d) Removing a detector.
   e) Modifying the placement a detector, since it can impact the application completion time by improving the routing.

Depending on the biochip fabrication technique, not all the moves are applicable. If subtractive fabrication is used (e.g., etching a printed circuit board), we apply only the moves for devices. If additive fabrication is used (e.g., inkjet printing), we are interested in a minimal electrode count, thus we apply both moves for the electrodes and the moves for the devices.

GenerateNeighborhood applies one by one all the moves defined above under the limits conditioned by the fabrication technique and by the execution of the biochemical assay (e.g., at least one reservoir for each input fluid). However, applying some of the moves can lead to revisiting solutions, and, consequently, to cycling between already evaluated architectures. To avoid this situation, such moves are considered tabu, and are stored in a tabu list. An example of a tabu move is adding a dispensing reservoir after having removed the same reservoir during the previous iteration. Hence, at each iteration, we apply only the moves that are not tabu, and we determine the tabu moves for the next iteration (line 6 in Algorithm 3).

Each of the architectures from the neighborhood $\mathcal{N}$ is evaluated using the following objective function:

$$\text{Objective}(\mathcal{A}) = \text{Cost}_{\mathcal{A}} + W \times \max\left(0, \delta_{\mathcal{G}}^k - D_{\mathcal{G}}\right) \quad (4)$$

where $\text{Cost}_{\mathcal{A}}$ is the cost of the architecture $\mathcal{A}$, calculate as presented in Section IV, and $\delta_{\mathcal{G}}^k$ is the completion time of the application $\mathcal{G}$ on $\mathcal{A}$ obtained with the proposed FA-LSR (see Algorithm 1). If $\mathcal{G}$ is schedulable, i.e., the timing constraints are met, the second term is 0, otherwise, we use a penalty weight W (a large constant) to penalize invalid architectures that are leading to unschedulable applications. The new solution $\mathcal{A}^{\text{current}}$ is obtained by selecting the architecture from $\mathcal{N}$ that minimizes the objective function (line 7 in Algorithm 3). If the currently found solution $\mathcal{A}^{\text{current}}$ is better than the best-so-far $\mathcal{A}^{\text{best}}$, then the latter is updated accordingly (lines 8–10).

In case the search does not find an architecture solution better than $\mathcal{A}^{\text{current}}$ for a number of iterations, then TS uses diversification (line 12). A diversification move, composed of two or more single-moves is applied on $\mathcal{A}^{\text{best}}$ in order to guide the search toward unexplored regions of the search space. For example, a diversification move was applied to the architecture from Fig. 12(a), resulting in the architecture from Fig. 12(e). The added electrodes are marked in Fig. 12(e) with a darker shade of gray. Next, the Restart function (line 43) updates, if necessary, the architecture $\mathcal{A}^{\text{best}}$ and the tabu list (deletes the previous elements and adds the tabu moves due to diversification). The search continues until the time limit is reached, when our TS-based architecture synthesis returns $\mathcal{A}^{\text{best}}$.

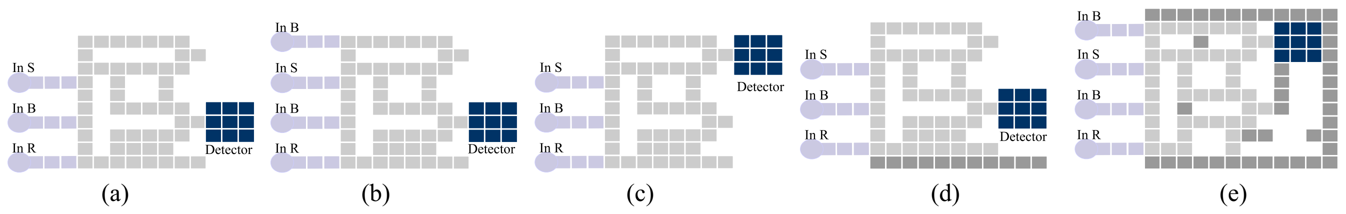


Fig. 12. Example of neighboring architectures. (a) $\mathcal{A}^{\text{current}}$. (b) Add reservoir for buffer. (c) Replace the detector. (d) Add bottom-row of electrodes. (e) Diversification move.

TABLE IV
COMPONENT LIBRARY

Name	Cost	Area (mm^2)	Time (s)
Electrode	1	1.5×1.5	N/A
Dispensing reservoir	3	2.5×2.5	2
Optical Detector	9	4.5×4.5	30

TABLE V
FLUIDIC LIBRARY

Fluid Name	Cost
Sample	1
Buffer	4
Reagent	10

TABLE VI
APPLICATION-SPECIFIC SYNTHESIS RESULTS OBTAINED BY TS

		$k = 0$			$k = 1$			$k = 2$		
App.	$d_{\mathcal{G}}$	Arch	C_R	C_{TS}	Arch	C_R	C_{TS}	Arch	C_R	C_{TS}
PCR	10	5×5 (2,1,1,0)	53	47	7×6 (1,1,1,0)	66	54	8×6 (1,1,1,0)	72	61
PDNA	60	5×5 (1,1,1,0)	49	38	6×5 (1,1,1,0)	54	40	6×5 (1,1,1,0)	54	49
IDP	200	5×5 (1,1,1,4)	85	74	5×5 (1,1,1,4)	90	84	5×5 (1,1,1,4)	90	84
SB_1	100	5×5 (1,1,1,3)	76	83	5×5 (1,1,1,4)	85	76	5×5 (1,1,1,4)	85	84

VII. EXPERIMENTAL RESULTS

For experiments we used four synthetic benchmarks ($SB_{1–4}$) [42] and five real-life applications: 1) the mixing stage of PCR (7 operations); 2) *in vitro* diagnostics on human physiological fluids (IVD, 28 operations); 3) the colorimetric protein assay (CPA, 103 operations); 4) the interpolation dilution of a protein (IDP, 71 operations); and 5) the sample preparation for plasmid DNA (PDNA, 19 operations). The application graphs for PCR, IVD, and CPA can be found in [33], for IDP in [43], and for PDNA in [14].[1] The algorithms were implemented in Java (JDK 1.6) and run on a MacBook Pro computer with Intel Core 2 Duo CPU at 2.53 GHz and 4 GB of RAM.

The focus of this paper is to determine if application-specific architectures are more cost-effective than rectangular architectures. Thus, we have used our TS-based approach to synthesize architectures for the PCR, PDNA, IDP, and SB_1 applications. Together with the results obtained by TS, we have also determined, using exhaustive search (which varies the architecture dimensions and the number of reservoirs and optical detectors), the cheapest general purpose architecture, which can run the application within the deadline. For a fair comparison, both exhaustive search and our TS-based architecture synthesis used the proposed FA-LSR for compilation. The CRM library $\mathcal{L}$ was determined using the algorithm in [25] for exhaustive search and the proposed ILB (see Section V) for the TS-based architecture synthesis. We have run the experiments for $k = 0$, 1 and 2 faults, estimating the operation execution time as proposed in Section V-A. We used the component library $\mathcal{M}$ and fluidic library $\mathcal{F}$ in Tables IV and V, respectively, and the parametric library determined in [24].

The results are presented in Table VI. The deadline $D_{\mathcal{G}}$ for each application is presented in column 2, the size of the rectangular architectures for $k = 0$, 1, and 2 are presented in columns 3, 6, and 9, respectively (the number in parentheses refer to the numbers of reservoirs for buffer, sample and reagent) and their cost C_R is in columns 4, 7, and 10. The results of TS for $k = 0$, 1, and 2 are presented in columns 5, 8, and 11, respectively. As we can see from Table VI, our TS is able to produce application-specific architectures which are significantly cheaper than the best general purpose architecture. For the PDNA application, our proposed synthesis obtained architectures that reduce the cost with 22.4%, 25.9%, and 9.2% for $k = 0$, 1, and 2, respectively. Our proposed methodology can also support the designer in performing a tradeoff between the yield and the cost of the architecture, by introducing redundant electrodes to tolerate permanent faults. The increase in cost for $k = 1$ and $k = 2$ is presented in columns 8 and 11 for TS. For the PCR application (see row 1 in Table VI), introducing redundancy for fault-tolerance resulted in a increase of 12.9% in the architecture cost.

In the second set of experiments, we compared the costs of the architecture obtained by our proposed TS-based synthesis to the cost of the architecture obtained using the SA-based synthesis, proposed in [10]. For a fair comparison, we used for cost calculation the method proposed in [10], which did not consider the cost of the input fluids. Since in this set of experiments we do not consider the optical detection operations, a limitation of [10], we have adjusted the deadlines to 10, 15, and 100 s for PCR, IVD, and CPA, respectively. The results are presented in Table VII. As we can see, our TS-based architecture synthesis is able to obtain better results. For example, for IVD (row 2 in Table VII), a reduction in cost of 28.3% was obtained using the TS-based synthesis proposed in this paper.

In the third set of experiments we were interested to determine the quality of the proposed compilation, FA-LSR (Section IV-C), in terms of the application completion time $\delta_{\mathcal{G}}$. We have compared $\delta_{\mathcal{G}}$ to the nearly-optimal $\delta_{\mathcal{G}}^{\text{opt}}$ obtained in [12] using TS for the compilation. Note that $\delta_{\mathcal{G}}$ is determined for the case when there are no faults, since the implementation in [12] does not consider faults. The results of

[1]We ignored the detection operations for the experiments presented in Tables VI, VII, and IX.

TABLE VII
COMPARISON OF TS AND SA [10]-BASED SYNTHESES

	$k=0$		$k=1$		$k=2$	
App.	C_{SA}	C_{TS}	C_{SA}	C_{TS}	C_{SA}	C_{TS}
PCR	60	43	65	46	98	78
IVD	62	56	70	62	85	78
CPA	59	49	66	63	127	123

TABLE VIII
EVALUATION OF THE FA-LSR COMPILATION (NO FAULTS)

App. (ops.)	Arch.	$\delta^0_{\mathcal{G}}(s)$	CPU time	$\delta^{opt}_{\mathcal{G}}(s)$	CPU time	Deviation (%)
PCR (7)	9×9	11	25 ms	10	60 min	9
IVD (28)	9×10	77	91 ms	73	60 min	5.4
CPA (103)	11×12	219	498 ms	214	60 min	2.3

TABLE IX
EVALUATION OF THE CRM APPROACH FOR COMPILATION

App. (ops.*)	Arch.	$\delta^R_{\mathcal{G}}(s)$	$\delta^{CRM}_{\mathcal{G}}(s)$	Deviation (%)
IVD (23)	45 (2,2,2)	18.4	11.73	36
SB$_3$ (50)	96 (1,2,1)	29.39	23.9	18.6
SB$_3$ (70)	103 (2,2,2)	31.03	20.15	35
SB$_4$ (90)	125 (2,2,2)	42.51	27.87	34

this comparison are presented in Table VIII. This comparison was only possible for rectangular architectures, a limitation of [12]. Also, for a fair comparison, we ignored routing and we have used the same module library as in [12].

The deadlines for PCR, IVD, and CPA are 10, 15, and 100 s, respectively. Table VIII shows that our LS-based compilation is able to obtain good quality results using a much shorter runtime (milliseconds versus 1 h). The average percentage deviation from the near-optimal result is 5.5%, hence, it can successfully be used for design space exploration.

In our final set of experiments we were interested to determine the efficiency of our proposed placement of operations (Section V) in terms of the application completion time $\delta^{\text{CRM}}_{\mathcal{G}}$ obtained after compilation. We compared $\delta^{\text{CRM}}_{\mathcal{G}}$ to the completion time $\delta^{R}_{\mathcal{G}}$, obtained using the routing-based compilation approach from [24], which is the only available compilation approach that is not limited to rectangular modules and can take advantage of an application-specific architecture. The results of this comparison are presented in Table IX. For the real-life application (IVD), we used the application-specific architecture (column 2) derived with our architecture synthesis from [10]. The application-specific architectures for the synthetic benchmarks were obtained manually. In column 2 we present, for each architecture, the number of electrodes and in parentheses the numbers of reservoirs for sample, buffer, and reagent. As we can see from Table IX, our placement results in a better completion time $\delta^{\text{CRM}}_{\mathcal{G}}$ (column 5) than $\delta^{R}_{\mathcal{G}}$ (column 4) for all the tested benchmarks. For example, for IVD, we obtained a completion time $\delta^{\text{CRM}}_{\mathcal{G}} = 11.73$ s, improving with 36% the completion time $\delta^{R}_{\mathcal{G}} = 18.4$ s.

VIII. CONCLUSION

We have proposed a TS-based synthesis approach for application-specific fault-tolerant DMB architecture, such that the architecture cost is minimized and the deadlines are satisfied even in case of permanent faults. The architecture alternatives, visited by TS, are evaluated in terms of their impact on the timing constraints of the application. We have proposed an LS-based compilation (FA-LSR) to determine the application completion time which depends on the given architecture and on the pattern of permanent faults. We have also proposed a strategy to incrementally build a library $\mathcal{L}$ of CRMs that take advantage of the characteristics of the architecture and use effectively the available area. We have proposed a method to estimate the operation execution in case of faults. The library $\mathcal{L}$ and the estimated operation execution times are used by FA-LSR to determine the application completion time in case of faults. As the experiments show, our LS-based compilation proves to be fast and provides good-enough solutions.

The experiments run on five real-life applications and four synthetic benchmarks show that our synthesis approach is able to significantly reduce the cost compared to general-purpose rectangular architectures. Hence, by synthesizing fault-tolerant architectures, our methodology can help the designer increase the yield of DMBs. We plan to further extend our architecture synthesis by considering faults that happen during the execution of the application. Hence, future work will integrate methods for detection and recovery from transient faults.

REFERENCES

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

[2] V. Srinivasan, V. K. Pamula, and R. B. Fair, "An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids," *Lab Chip*, vol. 4, no. 4, pp. 310–315, 2004.

[3] M. G. Pollack, A. D. Shenderov, and R. B. Fair, "Electrowetting-based actuation of droplets for integrated microfluidics," *Lab Chip*, vol. 2, no. 2, pp. 96–101, 2002.

[4] R. S. Sista *et al.*, "Digital microfluidic platform for multiplexing enzyme assays: Implications for lysosomal storage disease screening in newborns," *Clin. Chem.*, vol. 57, no. 10, pp. 1444–1451, 2011.

[5] F. Su and K. Chakrabarty, "Yield enhancement of reconfigurable microfluidics-based biochips using interstitial redundancy," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 2, pp. 104–128, 2006.

[6] T. Xu and K. Chakrabarty, "Fault modeling and functional test methods for digital microfluidic biochips," *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 4, pp. 241–253, Aug. 2009.

[7] P. Roy, H. Rahaman, C. Giri, and P. Dasgupta, "Modelling, detection and diagnosis of multiple faults in cross referencing DMFBs," in *Proc. Int. Conf. Informat. Electron. Vis.*, Dhaka, Bangladesh, 2012, pp. 1107–1112.

[8] E. Maftei, P. Pop, and J. Madsen, "Droplet-aware module-based synthesis for fault-tolerant digital microfluidic biochips," in *Proc. Symp. Design Test Integr. Packag. MEMS/MOEMS*, Cannes, France, 2012, pp. 47–52.

[9] Y. Zhao, K. Chakrabarty, R. Sturmer, and V. K. Pamula, "Optimization techniques for the synchronization of concurrent fluidic operations in pin-constrained digital microfluidic biochips," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 6, pp. 1132–1145, Jun. 2012.

[10] M. Alistar, P. Pop, and J. Madsen, "Application-specific fault-tolerant architecture synthesis for digital microfluidic biochips," in *Proc. 18th Asia South Pac. Design Autom. Conf.*, Yokohama, Japan, 2013, pp. 794–800.

[11] E. Maftei, P. Pop, J. Madsen, and T. K. Stidsen, "Placement-aware architectural synthesis of digital microfluidic biochips using ILP," in *Proc. Int. Conf. Very Large Scale Integr.*, 2008, pp. 425–430.

[12] E. Maftei, P. Pop, and J. Madsen, "Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices," *Design Autom. Embedded Syst.* vol. 14, no. 3, pp. 287–307, 2010.

[13] M. Alistar, P. Pop, and J. Madsen, "Online synthesis for error recovery in digital microfluidic biochips with operation variability," in *Proc. Symp. Design Test Integr. Packag. MEMS/MOEMS*, Cannes, France, 2012, pp. 53–58.

[14] Y. Luo, K. Chakrabarty, and T.-Y. Ho, "A cyberphysical synthesis approach for error recovery in digital microfluidic biochips," in *Proc. Conf. Design Autom. Test Europe*, Dresden, Germany, 2012, pp. 1239–1244.

[15] D. Grissom and P. Brisk, "Path scheduling on digital microfluidic biochips," in *Proc. 49th Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 26–35.

[16] D. Grissom and P. Brisk, "Fast online synthesis of generally programmable digital microfluidic biochips," in *Proc. 8th Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, New York, NY, USA, 2012, pp. 413–422.

[17] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation," *ACM J. Emerg. Technol. Comput. Syst.* vol. 3, no. 3, 2007, Art. no. 13.

[18] M. Cho and D. Z. Pan, "A high-performance droplet routing algorithm for digital microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1714–1724, Oct. 2008.

[19] T.-W. Huang, S.-Y. Yeh, and T.-Y. Ho, "A network-flow based pin-count aware routing algorithm for broadcast-addressing EWOD chips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 12, pp. 1786–1799, Dec. 2011.

[20] F. Su and K. Chakrabarty, "Module placement for fault-tolerant microfluidics-based biochips," *ACM Trans. Design Autom. Elect. Syst.*, vol. 11, no. 3, pp. 682–710, 2006.

[21] F. Su and K. Chakrabarty, "Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips," in *Proc. 42nd Annu. Design Autom. Conf.*, Anaheim, CA, USA, 2005, pp. 825–830.

[22] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design Test Comput.*, vol. 17, no. 1, pp. 68–83, Jan./Mar. 2000.

[23] T. Xu and K. Chakrabarty, "Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 4, no. 3, 2008, Art. no. 11.

[24] E. Maftei, P. Pop, and J. Madsen, "Routing-based synthesis of digital microfluidic biochips," *Design Autom. Embedded Syst.*, vol. 16, no. 1, pp. 19–44, 2012.

[25] M. Alistar, P. Pop, and J. Madsen, "Operation placement for application-specific digital microfluidic biochips," in *Proc. Symp. Design Test Integr. Packag. MEMS/MOEMS*, Barcelona, Spain, 2013, pp. 1–6.

[26] H. Ren and R. B. Fair, "Micro/nano liter droplet formation and dispensing by capacitance metering and electrowetting actuation," in *Proc. 2nd Conf. Nanotechnol.*, Washington, DC, USA, 2002, pp. 369–372.

[27] 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.*, vol. 39, no. 3, pp. 1153–1182, 2010.

[28] V. Srinivasan, V. K. Pamula, M. Pollack, and R. B. Fair, "A digital microfluidic biosensor for multianalyte detection," in *Proc. IEEE 16th Annu. Int. Conf. Micro Elect. Mech. Syst. (MEMS)*, Kyoto, Japan, 2003, pp. 327–330.

[29] E. Maftei, P. Pop, and J. Madsen, "Module-based synthesis of digital microfluidic biochips with droplet-aware operation execution," *ACM J. Emerg. Technol. Comput. Syst.* vol. 9, no. 1, 2013, Art. no. 2.

[30] Y. Luo, K. Chakrabarty, and T.-Y. Ho, "Design of cyberphysical digital microfluidic biochips under completion-time uncertainties in fluidic operations," in *Proc. 50th Annu. Design Autom. Conf.*, Austin, TX, USA, 2013, pp. 1–7.

[31] (2014). *Advanced Liquid Logic*. [Online]. Available: http://www.liquid-logic.com/technology.html.

[32] R. Sista *et al.*, "Development of a digital microfluidic platform for point of care testing," *Lab Chip*, vol. 8, no. 12, pp. 2091–2104, 2008.

[33] F. Su and K. Chakrabarty, "Benchmarks for digital microfluidic biochip design and synthesis," Dept. Elect. Comput. Eng., Duke Univ., Durham, NC, USA, Tech. Rep., Jan. 2006.

[34] J.-D. Huang, C.-H. Liu, and T.-W. Chiang, "Reactant minimization during sample preparation on digital microfluidic biochips using skewed mixing trees," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2012, pp. 377–383.

[35] S. Even, "An algorithm for determining whether the connectivity of a graph is at least *k*," *SIAM J. Comput.*, vol. 4, no. 3, pp. 393–396, 1975.

[36] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems," in *Proc. Conf. Design Autom. Test Europe*, Munich, Germany, 2006, pp. 1–6.

[37] O. Sinnen, *Task Scheduling for Parallel Systems*. Hoboken, NJ, USA: Wiley, 2007.

[38] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. River Edge, NJ, USA: World Sci., 1999.

[39] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *J. Glob. Optim.*, vol. 6, no. 2, pp. 109–133, 1995.

[40] P. Paik, V. K. Pamula, and R. B. Fair, "Rapid droplet mixers for digital microfluidic systems," *Lab Chip*, vol. 3, no. 4, pp. 253–259, 2003.

[41] F. Glover and M. Laguna, *Tabu Search*. Boston, MA, USA: Kluwer, 1997.

[42] E. Maftei, "Synthesis of digital microfluidic biochips with reconfigurable operation execution," Ph.D. dissertation, Dept. Inform. Math. Model., Sect. Embedded Syst. Eng., Tech. Univ. Denmark, Kongens Lyngby, Denmark, 2011.

[43] Y. Zhao, T. Xu, and K. Chakrabarty, "Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 6, no. 3, 2010, Art. no. 11.

Mirela Alistar received the Ph.D. degree in computer engineering from the Technical University of Denmark, Kongens Lyngby, Denmark, in 2014.

Since 2010, she has been researching on the development of computer-aided design tools for modeling and control optimization of digital microfluidic device. She is currently a Post-Doctoral Fellow with Hasso Plattner Institute, Potsdam, Germany. She is supporting open access research and has organized citizen-science events, where she dissiminates to the public with the aim of involving them into creating more knowledge. Her current research interests include system-level design of embedded systems, with a special focus on cyber physical systems.

Paul Pop received the Ph.D. degree in computer systems from Linköping University, Linköping, Sweden, in 2003.

He is an Associate Professor with the DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark. At DTU Compute, he coordinates a research group focused on safety-critical embedded systems. His current research interests include system-level design of embedded systems. He has published extensively in the above area, over 100 peer-reviewed international publications, three books, and seven book chapters.

Dr. Pop was a recipient of the Best Paper Award at the Design, Automation and Test in Europe Conference in 2005 and the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems in 2010.

Jan Madsen received the Ph.D. degree in computer science from the Technical University of Denmark, Kongens Lyngby, Denmark, in 1992.

He is a Full Professor of Computer-Based Systems with the DTU Compute, Technical University of Denmark. He holds two patents, from which he has co-founded Biomicore, Frederiksberg, Denmark. He is the Deputy Director of the Department and the Head of the Embedded Systems Engineering section with DTU Compute. He has published over 140 peer-reviewed conference and journal papers, 12 book chapters, one book, and four edited books. His current research interests include methods and tools for systems engineering of computing systems, embedded systems-on-a-chip, wireless sensor networks (Internet-of-things), microfluidic labs-on-a-chip, synthetic biology, and design, modeling, analysis, and optimization of such systems.

Dr. Madsen was a recipient of several best paper nominations, two Best Paper Award at MECO 2013 and CASES 2009, one paper among the 30 most influential papers from ten years of Design Automation and Test in Europe, and three papers among the highly cited papers in System Codesign and Synthesis.