Application-Specific Fault-Tolerant Architecture Synthesis for Digital Microfluidic Biochips

Mirela Alistar Paul Pop Jan Madsen

Department of Informatics and Mathematical Modeling
Technical University of Denmark, Denmark
Tel: +45-45253470 Fax: 45-4593-0074
e-mail: {mali, pop, jan}@imm.dtu.dk

***Abstract*—Microfluidic-based biochips are replacing the conventional biochemical analyzers, and are able to integrate on-chip all the necessary functions for biochemical analysis using microfluidics. The digital microfluidic biochips are based on the manipulation of liquids not as a continuous flow, but as discrete droplets on an array of electrodes. Microfluidic operations, such as transport, mixing, split, are performed on this array by routing the corresponding droplets on a series of electrodes. Researchers have proposed several approaches for the synthesis of digital microfluidic biochips. All previous work assumes that the biochip architecture is given, and most approaches consider a rectangular shape for the electrode array. However, non-regular application-specific architectures are common in practice. Hence, in this paper, we propose an approach to the application-specific architecture synthesis. Our approach can also help the designer to increase the yield by introducing redundant electrodes to tolerate permanent faults. The proposed architecture synthesis algorithm has been evaluated using several benchmarks.**

I. Introduction

Microfluidic biochips have the potential to replace the conventional laboratory equipment as they integrate all the functions needed to complete a bioassay. Applications on biochips are considered in areas such as drug discovery, clinical diagnosis, DNA sequencing, protein analysis and immunoassays [1], [2]. The digital microfluidic biochips (DMBs) use discrete amounts of fluids of nanoliter volume, named droplets, to perform operations such as: dispensing, transport, mixing, split, dilution and detection.

A DMB is modeled as an array of identical electrodes, see Fig. 1a, where each electrode can hold a droplet. Operations such as mix and dilution are reconfigurable, i.e., they may take place on any of the electrodes in the array. The reconfigurable operations "execute" on "virtual devices" (also called modules) and droplets are transported on "virtual" routes, since such execution and transport can take place on any of the electrodes.

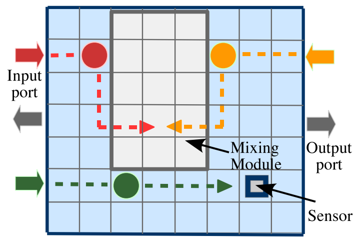
There is a significant amount of work on the synthesis of DMBs [1], [3], [4], which consists of the following tasks: allocation, binding, scheduling, placement and routing. The output of these synthesis tasks is the "electrode actuation sequence", which controls the movement of droplets to run the biochemical application. We will call these synthesis tasks *compilation*, to distinguish it from the architecture *synthesis* proposed in this paper.

Also, we make the distinction between the *physical* biochip architecture, consisting of *physical components*, such as electrodes, input and output reservoirs and detectors, and *virtual* devices which are allocated and placed on the physical array of electrodes and interconnected using *virtual* routes. All previous work assumes that the physical architecture is given, and most researchers use general-purpose architectures, which have a rectangular shape (Fig. 1a). However, in practice, application-specific architectures which are non-regular (Fig. 3) are more common because they can significantly reduce the costs by reducing the number of components used in the physical architecture. For example, application-specific architectures have been proposed for disease screening in newborns [5] and for diagnostics on human physiological fluids [2].

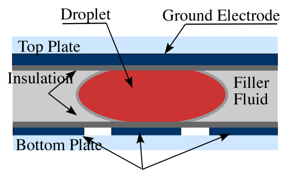
Therefore, in this paper we propose an approach to the architecture synthesis of application-specific biochips (see Fig. 6). Starting from a biochemical application, modeled as a sequencing graph, and a library of physical components, our synthesis decides a physical biochip architecture that minimizes unit costs and can execute the application within its specified deadline. Fig. 3 shows such an architecture example for the application in Fig. 2. Our synthesis approach decides the allocation of physical components (electrodes, input and output reservoirs, detectors), their placement and interconnection. However, our synthesis does not decide the allocation and placement of virtual devices, or the virtual routes, since these can use any of the physical electrodes on the architecture. We assume that these decisions, together with binding and scheduling of the operations in the biochemical application are performed using existing synthesis approaches [1], taking as input the non-regular application-specific physical architecture derived by our algorithm.

Yield is a big concern for biochips, hence researchers have proposed fabrication methodologies to increase the yield of DMBs, e.g., from a very low 30% to 90% [6]. DMBs are affected by permanent faults (e.g., due to abnormal layer deposition, short between two adjacent electrodes, see [7] for more details), which may lead to the failure of the biochemical application. Hence, biochips are tested after fabrication [7]. If permanent faults are detected, the biochip is discarded, unless the applications can be reconfigured to avoid them [8]. In order to increase the yield, which is very important for the market success of DMBs, our physical architecture synthesis approach can introduce redundancy (e.g., extra electrodes) to tolerate a given number of permanent faults.

To the best of our knowledge, this is the first time when an approach to the synthesis of fault-tolerant application-specific architectures has been proposed. Researchers have so far only considered varying the dimensions of purely rectangular general-purpose architectures or have addressed limited aspects such as minimizing the number of pins used to control the electrodes [9], which is orthogonal to our problem. Also, the issue of fault-tolerance has only been tackled in the context of rectangular architectures, by introducing a regular pattern of redundant electrodes [8].



(a) General-purpose biochip



(b) Droplet movement

Fig. 1: Biochip architecture model

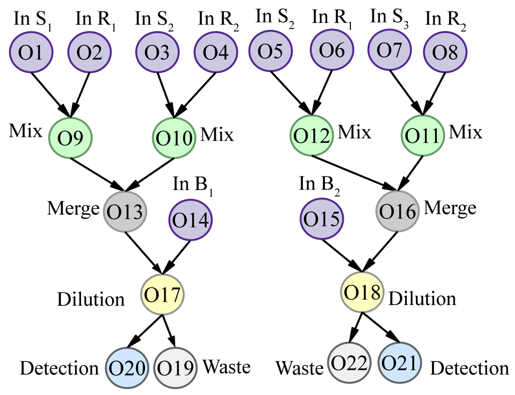


Fig. 2: Biochip application model

The biochip architecture and application models are presented in the next section. The synthesis problem is presented in Section III using a motivational example. We have proposed a heuristic approach based on Simulated Annealing (SA) to solve this optimization problem (Section V). Each architecture solution visited by SA is evaluated in terms of its cost, performance and fault-tolerance to permanent faults, i.e., regardless of where permanent faults happen, the application can be reconfigured to complete within its deadline. We have proposed a List Scheduling (LS)-based heuristic, which determines the worst-case completion time of an application in case of faults, see Section IV. The proposed synthesis methodology has been evaluated in Section VI using several benchmarks.

II. System Model

A. Biochip Architecture

In a DMB, a droplet is sandwiched between a top ground electrode and a bottom electrode as shown in Fig. 1b. Two glass plates, a top and a bottom one, protect the DMB from external factors. The droplets are manipulated using the electrowetting-on-dielectric (EWOD) principle. For example, in Fig. 1b, if the middle electrode on the bottom plate is turned off, and the left electrode is activated by applying voltage, the droplet will move to the left.

There are two types of operations: (i) non-reconfigurable, such as dispensing, executed on input reservoirs, detection, using on-chip optical detectors, and (ii) reconfigurable, such as droplet routing, mixing, dilution and split operations. A mixing operation is executed when two droplets are moved to the same location and then transported together. A split operation is done by applying concurrently the same voltage on both left and right electrodes, while the middle one remains turned off. Dilution is a mixing operation followed by a split operation. The reconfigurable operations are executed using the physical electrodes in the biochip architecture array.

Researchers have used several approaches to group these electrodes and thus form "virtual devices" on which the operations execute [10]. The straightforward approach used by most researchers is to consider a rectangular area of electrodes, called a "module". However, an operation can execute anywhere on the microfluidic array, and it is not necessarily confined to a rectangular area, as is the case with the routing-synthesis approach [10], which allows operations to execute on any route. In this paper, we consider the approach from [11] where the virtual devices have rectangular shapes, but the position of droplets inside these modules is known, hence the name, "droplet-aware operation execution", as opposed to black-box module-based operation execution, which ignores the positions of droplets. The advantage of knowing the position of the droplets inside the module is that we can control the droplets movement to avoid accidental droplet merging with adjacent modules (black-box modules are typically surrounded by a border of unused electrodes to avoid such situations), or with routes (which had to be routed around the black-box modules).

Although the approach used for operation execution on virtual devices is orthogonal to our architecture synthesis methodology, we have decided to use the "droplet-aware" approach [11] because of its improved reconfigurability in case of permanent faults: the droplets are simply instructed to avoid the faulty electrodes.

Based on experiments, researchers characterize a virtual device library $\mathcal{L}$, such as Table I, which provides the shape and corresponding execution time that are needed for each operation. For example, as seen in Table I, the execution time for a mixing operation on a 3×6 rectangular module is 2.52 s if no faults are present. Section IV-B presents a way to determine, at design time, the worst-case operation execution in case of k permanent faults.

Fig. 3 shows an application-specific architecture $\mathcal{A}$. The cost of an architecture depends on the costs of the components, which are provided by the designer in a physical components library $\mathcal{M}$ (Table II) and is defined as:

$$Cost_{\mathcal{A}} = \sum N_{M_i} \times Cost_{M_i} \quad (1)$$

, where N_{M_i} is the number of components of type M_i and $Cost_{M_i}$ is the cost of the *physical* component M_i from the library $\mathcal{M}$. The total cost of $\mathcal{A}$ from Fig. 3 is 187 units.

TABLE I: Virtual Device Library

Op.	Shape	Time (s) no faults	Time (s) $k=1$	Time (s) $k=2$
Mix	3×6	2.52	2.71	3.77
Mix	5×8	2.05	2.09	2.3
Mix	4×7	2.14	2.39	2.51
Mix	5×5	2.19	2.28	2.71
Mix	$5\times5\times1$	2.19	2.73	3.92
Mix	$5\times5\times2$	3.98	5.82	7.56
Dilution	3×6	4.4	4.67	4.11
Dilution	5×8	3.75	4.76	6.3
Dilution	4×7	3.88	4.22	4.46
Dilution	5×5	3.98	4.12	4.67
Split	1×1	0	0	0
Storage	1×1	N/A	N/A	N/A

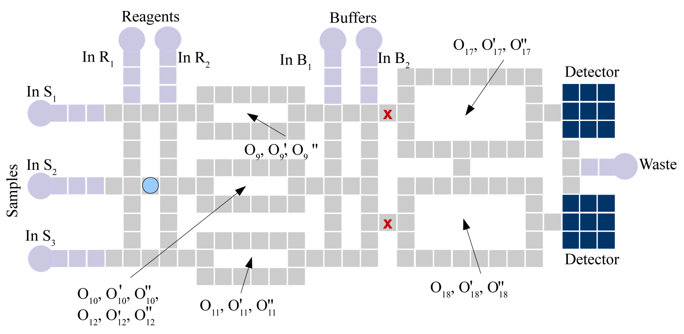

Reagents
Buffers
In R$_1$
In R$_2$
In B$_1$
In B$_2$
O$_{17}$, O'$_{17}$, O''$_{17}$
Detector
In S$_1$
O$_9$, O$_9$', O$_9$ ''
In S$_2$
Waste
Samples
In S$_3$
Detector
O$_{10}$, O'$_{10}$, O''$_{10}$,
O$_{12}$, O'$_{12}$, O''$_{12}$
O$_{11}$, O'$_{11}$, O''$_{11}$
O$_{18}$, O'$_{18}$, O''$_{18}$

Fig. 3: Application-specific biochip architecture

B. Biochemical Application Model

A biochemical application is modeled using an acyclic directed graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where the nodes $\mathcal{V}$ represent the operations, and the edges $\mathcal{E}$ represent the dependencies between them. Every operation O_i has an associated execution time C_i, given in the library $\mathcal{L}$ (Table I), for reconfigurable operations, and in library $\mathcal{M}$ (Table II) for non-reconfigurable operations. In this paper, we consider that routing a droplet from one electrode to another takes 0.01 s. In addition, the application $\mathcal{G}$ has to complete by a given deadline $D_{\mathcal{G}}$. Let us consider the biochemical application graph from Fig. 2, which has 22 operations. The directed edge between O_{17} and O_{20} signifies that operation O_{17} has to finish before operation O_{20} can start executing. Operation O_{20} uses the output droplet issued by dilution 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. 3.

III. Problem Formulation

In this paper we address the following problem. Given as input a biochemical application modeled as a graph, the libraries $\mathcal{M}$ and $\mathcal{L}$ and a number k of permanent faults that have 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 is within the deadline $D_{\mathcal{G}}$ for any occurrence of the k faults.

A. Motivational Example

Let us consider an application graph $\mathcal{G}$ obtained by repeating three times (due to space reasons) the graph from Fig. 2. We are interested to synthesize a physical architecture for this application, considering the component library $\mathcal{M}$ from Table II, such that the cost is minimized and a deadline of $D_{\mathcal{G}}$ = 20 s is satisfied (we ignore the detection operations for this example). After an architecture is synthesized, we compile the application $\mathcal{G}$ such that, using the virtual device library $\mathcal{L}$ from Table I, we obtain the completion time $\delta_{\mathcal{G}}$.

So far, researchers have considered only general-purpose biochips of $n \times m$ rectangular shape. To complete $\mathcal{G}$ within deadline $D_{\mathcal{G}}$ using a rectangular architecture $\mathcal{A}_2$, we need an array of 19×9 electrodes and eight reservoirs: two for the reagent, two for the buffer, three for the sample and one for the waste. In this case the optimal completion time is $\delta_{\mathcal{G}}$ = 18.49 s, smaller than $D_{\mathcal{G}}$ = 20 s, and the architecture cost is $Cost_{\mathcal{A}_2}$ = 213. However, this costs can be reduced if we use an application-specific architecture $\mathcal{A}_1$, such as the one in Fig. 3,

TABLE II: Component Library

Name	Unit cost	Dimensions (mm)	Time (s)
Electrode	1	1.5×1.5	N/A
Input Reservoir	3	1.5×4.5	2
Waste Reservoir	3	1.5×4.5	N/A
Capacitive Sensor	1	1.5×4.5	0
Optical Detector	9	4.5×4.5	8

with a cost of 187. The completion[1] time on this architecture is $\delta_{\mathcal{G}}$ = 18.98 s, within the deadline $D_{\mathcal{G}}$. 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 still completes within the deadline. For our example, the completion time increases to $\delta_{\mathcal{G}}^{k=1}$ = 19.41 s in the worst-case.

We assume that our architecture synthesis, outlined in Fig. 6, is part of a methodology which has the following steps: (1) we synthesize an application-specific architecture $\mathcal{A}$ for an application $\mathcal{G}$; (2) we fabricate the biochips with this architecture; (3) all the biochips are tested to determine if they have permanent faults, and their location [7]; (4) if there are more than k faults, the chip is discarded, otherwise we perform a compilation of $\mathcal{G}$ on $\mathcal{A}$ (excluding the faulty electrodes) to obtain the electrode actuation sequence, using the approach from [8].

The architecture synthesis problem presented in this paper is NP-complete. We have proposed a metaheuristic approach based on Simulated Annealing (SA) to solve this optimization problem, presented in Section V. Each architecture solution visited by SA has to be evaluated in terms of its unit cost and checked if it meets the deadline even in the worst-case fault-occurrence (the exact cost function is presented in Section V). The next section presents our proposed architecture evaluation approach to derive the worst-case application completion time $\delta_{\mathcal{G}}^{k}$, considering an application $\mathcal{G}$, an architecture $\mathcal{A}$ and k permanent faults. Note that since the architecture synthesis is performed before the fabrication and testing, we cannot know the permanent faults when we do the architectural evaluation.

IV. Architecture Evaluation

To find out if an architecture $\mathcal{A}$ can run an application $\mathcal{G}$ within the deadline $D_{\mathcal{G}}$, we need to determine the application completion time $\delta_{\mathcal{G}}$. The value of $\delta_{\mathcal{G}}$ is obtained through a compilation process that has five steps: (i) *allocation*, which selects the devices to be used from the virtual devices library $\mathcal{L}$; (ii) *binding* the selected modules to the operations in the application $\mathcal{G}$; (iii) *placement*, which decides the positions of the virtual devices on the biochip architecture $\mathcal{A}$; (iv) *scheduling*, which decides the order of the operations; and (v) *routing*, which determines the droplet routes to bring the droplets to the needed locations on the biochip.

This compilation process is a NP-complete problem for which several approaches have been proposed. Metaheuristics, such as [1], [4] are able to obtain near-optimal results for $\delta_{\mathcal{G}}$, but take a long time. Recently, LS-based heuristics [12] have been proposed, which are faster, but cannot guarantee the optimality. In order to avoid discarding valid architectures (i.e., for which the optimal $\delta_{\mathcal{G}}^{opt} \le D_{\mathcal{G}}$), we need during the architecture evaluation a value of $\delta_{\mathcal{G}}$ that is as close as possible

[1]The binding of operations in the application are shown in the figure; we replicate 3 times the graph in Fig. 2, hence for every O_i, we have O_i' and O_i''.

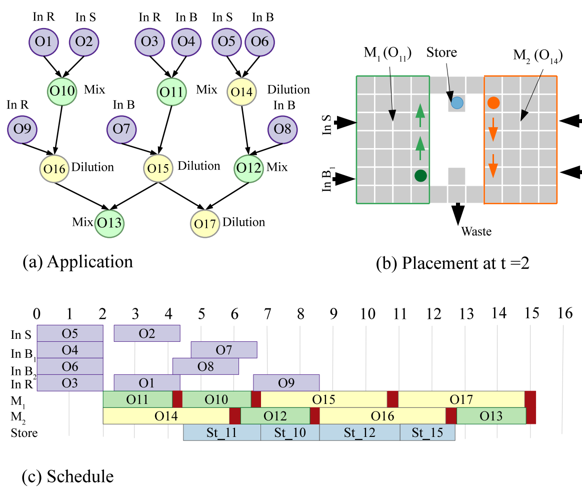


Fig. 4: Compilation example

to the optimal value. However, the calculation of $\delta_{\mathcal{G}}$ should be very fast, so it can be used for the evaluation of alternative solutions during the design space search performed by SA.

Hence, in this paper, we extend the LS-based compilation heuristic from [12] to be used for fault-tolerant architecture evaluation (Section IV-C). The main extension is concerned with considering the k permanent faults. As mentioned, we do not know the position of the k faults during the architecture synthesis (they will be known after fabrication and testing), so our evaluation has to determine the $\delta_{\mathcal{G}}^k$ in the worst-case. This worst-case $\delta_{\mathcal{G}}^k$ application completion time in case of k faults should be safe, i.e., not smaller than the exact worst-case (otherwise it may lead to synthesizing invalid architectures). It should also not be too pessimistic (i.e., much larger than the exact worst-case), because then it could discard many valid architectures.

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 [13]. 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 the virtual devices that can be affected by these faults. Instead, we take a potentially pessimistic but simpler approach, where we consider that each device in the application graph $\mathcal{G}$ could suffer from k faulty electrodes, and we have proposed, in Section IV-B, a technique to determine the overhead of k faults on an operation execution. These updated operation execution values C_i^k are then used inside our LS-based heuristic to determine a pessimistic (but safe) value for $\delta_{\mathcal{G}}^k$.

One of the five steps of the compilation process is routing. Due to permanent faults, an architecture can become disconnected, as in the case of the biochip in Fig. 3, considering the two permanent faults marked with a red “×”. If a biochip architecture under evaluation can be disconnected using k faults, it should be discarded and in this case the evaluation of $\delta_{\mathcal{G}}^k$ is no longer meaningful. Therefore, we introduce a routability check, described in the next subsection, which is applied before the completion time evaluation.

Let us illustrate the tasks that any compilation technique has to perform in order to obtain $\delta_{\mathcal{G}}^k$. Let us assume that we have to compile the graph from Fig. 4a on the application-specific biochip from Fig. 4b, using the virtual devices library $\mathcal{L}$ from Table I. We consider $k = 1$ and we calculate the worst-case increase in the operation execution C_i using the approach from Section IV-B, resulting in the execution times $C_i^{k=1}$ in column 4 of Table I. Next, let us assume that, at time $t = 2$ s mixing operation O_{11} has the highest priority among all the ready operations (an operation is ready if all its input droplets have arrived). Out of the allocated virtual devices for O_{11} from the library (Table I), the operation O_{11} is bound to 4×7 virtual device M_1, since it can be placed on the biochip and has the fastest completion time. Module M_1 is placed on the biochip as in Fig. 4b. At time $t = 4.14$ s, the operation O_{11} finishes executing, and its successor operation O_{15} becomes ready to execute. The schedule is depicted in Fig. 4c, as a Gantt chart, where the operations are presented as rectangles with the length equal to their duration, measured in seconds. For each operation we use the worst-case execution time $C_i^{k=1}$, which considers the execution delay in case of $k = 1$ faults (marked with red in Fig. 4c). Note that at $t = 4.14$ s, operation O_{11} finishes, but its output droplet cannot be used immediately, so it has to be stored on the biochip (see *Store* location in Fig. 4b). The total completion time for the application is 15.22 s. For simplicity reasons, we ignored routing in this example.

A. Routability with Permanent Faults

We want to guarantee that the architecture solution can run the application regardless of the location of k permanent faults. We say that an architecture passes the fault-tolerant routability test, if, in any scenario of k permanent faults, there is at least one route that connects each non-faulty electrode to the other non-faulty electrodes. We used the polynomial time $O(kn^3)$ algorithm from [14], that tests the k-vertex connectivity of a graph, to check the fault-tolerant routability of an architecture. For this purpose we model the architecture as a graph, in which the nodes represent the electrodes and the edges represent the direct connection between them. Note that 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 [14] tests if the graph remains connected in case of removal of k nodes. For example, the architecture in Fig. 3 is still connected for $k = 1$, but becomes disconnected for $k = 2$, e.g., if the 2 faults happen as indicated with the red “×”.

B. Operation Execution Overhead

Considering a virtual device M_i executing an operation O_i within C_i seconds, we are interested to determine the

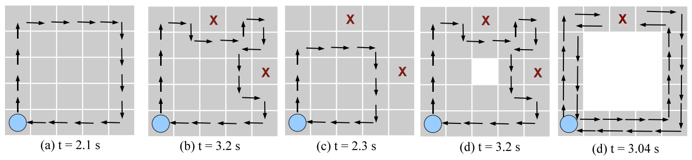


Fig. 5: Fault-tolerant operation execution

execution time C_i^k in case the virtual device M_i is placed over an area which has k faulty electrodes. We use the "droplet-aware" operation execution approach [11], outlined in Section II-A, which knows the positions of the droplets inside a virtual device. Let us consider the mixing example in Fig. 5a, performed on a 5×5 virtual device[1]. During fault-free operation execution, the droplet will move on that pattern, which leads to the fastest mixing time (the pattern depicted in Fig. 5a with arrows, and corresponding to column 3 in Table I). Let us suppose that after fabrication and testing (step 4 in our overall methodology, see Section III-A) we have detected two permanent faults, and after compilation, the virtual device is placed over these affected electrodes as presented in Fig. 5b. Because we use a "droplet-aware" operation execution, we can instruct the droplet, which normally moves as indicated in Fig. 5a, to avoid the faulty electrodes. However, this will increase the operation execution times, since the droplet no longer moves on its optimal path, with respect to the mixing time. We can use the "routing-based synthesis" approach from [10], which characterizes the operation completion time on a given route, to determine the overhead on C_i due to the change in route to avoid the two faults.

However, during the architecture synthesis (step 2 in our methodology) we do not know the position of the faults, hence, our approach is to determine the worst-case execution time C_i^k, i.e., the largest operation execution time among all possible combinations of k faults placed on the electrode of the virtual device M_i. We assume that the designer will provide such a C_i^k for every virtual device in Table I, for that particular k number of faults which has to be tolerated.

The value of C_i^k has to be determined only once, when the virtual devices library $\mathcal{L}$ is characterized. Because the virtual devices have a small area, the designer could use exhaustive search to determine, for each possible combination[2] of k faults on the device's electrodes, the best new route which avoids the faults, and leads to the fastest operation execution. The largest time among these is C_i^k. For example, for the device in Fig. 5a, the route in Fig. 5c is better (in this case, optimal), compared to the route in Fig. 5b (which has too many $90°$ turns, where the amount of mixing is limited, see [10] for details). Columns 4 and 5 in Table I present the values of C_i^k for k=1 and 2, respectively. We have also added to $\mathcal{L}$ in Table I (rows 5, 6) modules with margins of 1 and 2 electrodes, and empty inside.

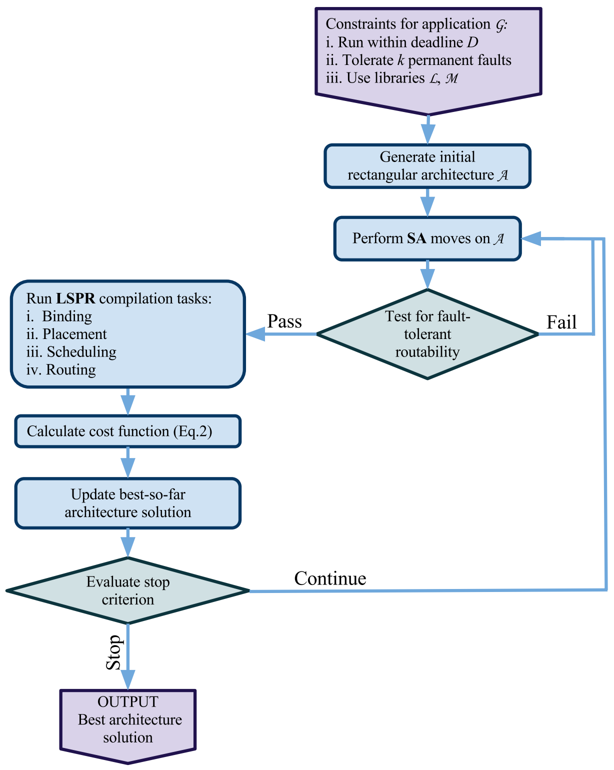
C. Biochemical Application Compilation

We perform a compilation of the biochemical application $\mathcal{G}$ on the architecture under evaluation $\mathcal{A}$ to determine the worst-case completion time $\delta_{\mathcal{G}}^k$ in case of k permanent faults. We use a List Scheduling (LS)-based heuristic to perform the binding and scheduling of the operations in $\mathcal{G}$. During scheduling, we also perform placement and routing.

Our LS-based heuristic, called LSPR (List Scheduling, Placement and Routing) takes as input the application graph $\mathcal{G}$, the biochip architecture $\mathcal{A}$, the virtual devices library $\mathcal{L}$, and the number of permanent faults k to be tolerated, and outputs the worst-case completion time $\delta_{\mathcal{G}}^k$.

Every node from $\mathcal{G}$ is assigned a specific priority according to the critical path priority function [13]. An operation O_i is ready to run when all its input droplets have arrived. 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 virtual devices library $\mathcal{L}$, to find the virtual device M_i that can be placed and scheduled at the earliest time. O_i is bound to M_i and scheduled at the corresponding time. For O_i, we use the worst-case operation time[3] C_i^k from $\mathcal{L}$, as discussed in the previous subsection. A dispensing operation, such as O_1 in Fig. 2, has no predecessor operations, therefore, if the corresponding reservoir is available, it can be scheduled at time $t = 0$. However, until they can be used, the dispensed droplets have to be stored on the biochip, occupying areas that can be used for other operations. To avoid this situation, we schedule the dispensing operations only when the dispensed droplets are needed. When a scheduled operation finished executing, the ready operations list is updated with the operations that have become ready. LSPR terminates when the ready operations list is empty.

An operation can be scheduled when it can be placed on the biochip. We have adapted the Fast Template Placement (FTP) algorithm from [15], which uses: (i) free-space partitioning manager that divides the free space in maximal empty rectangles (MERs) and (ii) a search engine that selects the



Constraints for application $\mathcal{G}$:
i. Run within deadline D
ii. Tolerate k permanent faults
iii. Use libraries $\mathcal{L}$, $\mathcal{M}$
Generate initial rectangular architecture $\mathcal{A}$
Perform **SA** moves on $\mathcal{A}$
Test for fault-tolerant routability
Pass
Fail
Run **LSPR** compilation tasks:
i. Binding
ii. Placement
iii. Scheduling
iv. Routing
Calculate cost function (Eq.2)
Update best-so-far architecture solution
Evaluate stop criterion
Continue
Stop
OUTPUT Best architecture solution

Fig. 6: Architecture synthesis flowchart

[1]We also handle irregular-shaped devices, with empty areas inside.

[2]We have implemented such an exhaustive search. The worst-case increase in C_i^k for $k = 1$ and 2, in each module, can be seen in Table I by comparing columns 4 and 5 with column 3.

[3]Even with this pessimistic approach, the increase in $\delta_{\mathcal{G}}$ is only 10%, see Table IV, which we believe is acceptable when performing architecture evaluation inside a design space exploration loop. After the chip is manufactured, and the faults are identified, we run the compilation from [8] (takes 1 hour), which has no pessimism because it knows the fault locations.

best-fit rectangle for each virtual device. FTP takes as input the virtual device M that needs to be placed on the biochip architecture $\mathcal{A}$ and the list of MERs L_{rect}. The search engine evaluates all MERs from L_{rect} that can accommodate M, and selects the one which is the nearest to the bottom-left corner of the biochip. We have adapted FTP for application-specific architectures such that we can place virtual devices of irregular shape and which may have missing electrodes, see Fig. 3.

We also need to determine the routes of the droplets between the virtual devices. In case of the black-box approach (see Section II-A), the droplets have to avoid the virtual devices, thus the route of a droplet can be obstructed by a virtual device. However, by using the "droplet-aware" approach, we can allow the droplets to pass through the virtual devices. In our evaluation of $\delta^k_{\mathcal{G}}$ we are not interested in the actual routes, only in their length. Hence, we have adapted the "filling phase" of Hadlock's algorithm [16] to determine the route lengths, considering the missing electrodes in the array (gaps) as the obstacles to be avoided, and including the worst-case overhead for the detours needed to avoid k faults.

V. Architecture Synthesis

We use a Simulated Annealing (SA) [17] metaheuristic for our application-specific architecture synthesis optimization problem (see Fig. 2). Our focus is to determine if application-specific architectures are more cost effective than the rectangular architectures used so far. We have chosen SA because it is easy to implement. In the future, we will investigate if other metaheuristics, such as Tabu Search or Evolutionary Algorithms are better suited to tackle our optimization problem. In general, the consensus is that the choice of heuristic depends on the nature of the problem, and none is always superior over the others.

SA 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 virtual devices library $\mathcal{L}$ and produces that architecture $\mathcal{A}$, which minimizes the objective function:

$$Objective(\mathcal{A}) = Cost_{\mathcal{A}} + W \times max(0, \delta^k_{\mathcal{G}} - D_{\mathcal{G}}) \qquad (2)$$

,where $Cost_{\mathcal{A}}$ is the cost of the architecture $\mathcal{A}$ currently evaluated and $\delta^k_{\mathcal{G}}$ is the worst-case completion time of the application $\mathcal{G}$ on $\mathcal{A}$ obtained with our LSPR algorithm. If $\mathcal{G}$ is schedulable, the second term is 0, otherwise, we use a penalty weight W (a large constant) to penalize invalid architectures (leading to unschedulable applications). Thus, we allow SA to explore invalid solutions, in the hope to guide the search towards valid architectures.

SA explores the solution space using design transformations called moves. A new solution, obtained by performing moves on the current solution, is accepted if it is an improved one. However, SA also accepts worse solutions, with a probability that depends on the objective function and the control parameter called *temperature*. We use the following moves to explore the design space: (1) adding/removing and changing the placement for non-reconfigurable components and (2) adding and removing reconfigurable components. (1) We increase and decrease the number of non-reconfigurable components, such as input reservoirs, detectors and sensors, under the limits conditioned by the execution of the biochemical assay (i.e., for example, we have at least one reservoir for each type of substance). In case detectors are used, we also modify their placement, since it can impact the completion time of the bioassay by improving the routing times. (2) The move on the reconfigurable components is performed by adding/removing electrodes. We distinguish between two cases: (i) adding/removing a single electrode and (ii) adding/removing a group of electrodes on the sides of the architecture. For each biochemical application, we calibrated the cooling schedule, defined by initial temperature TI , temperature length TL and cooling ratio ε. The algorithm stops when the *temperature* is cooled down to 1.

TABLE III: Architecture evaluation pessimism (no faults)

App. (ops.)	Arch.	$\delta^0_{\mathcal{G}}(s)$	Exec. time	$\delta^{opt}_{\mathcal{G}}(s)$	Exec. 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 IV: Architecture evaluation pessimism ($k = 0, 1, 2$)

App.	Cost	$\delta^0_{\mathcal{G}}$ (s)	$\delta^1_{\mathcal{G}}$ (s)	Deviation (%)	$\delta^2_{\mathcal{G}}$ (s)	Deviation (%)
PCR	98	8.42	8.81	4.6	9.43	11.9
IVD	85	12.62	13.11	3.8	14.81	17.3
CPA	129	153.9	169.3	10	190.11	23.5

VI. Experimental Results

For experiments we used three real-life applications [1]: (1) the mixing stage of polymerase chain reaction (PCR, 7 ops.); (2) in-vitro diagnostics on human physiological fluids (IVD, 28 ops.); (3) the colorimetric protein assay (CPA, 103 ops.)(We ignored the detection operations.) The deadlines for the applications are 10 s, 15 s and 100 s, respectively. 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. We used the $\mathcal{L}$ and $\mathcal{M}$ libraries from Table I and Table II.

In the first set of experiments we were interested to determine the pessimism of LSPR (Section IV-C) in terms of the completion time $\delta^k_{\mathcal{G}}$. We have first compared $\delta^0_{\mathcal{G}}$ to the nearly-optimal $\delta^{opt}_{\mathcal{G}}$ obtained in [4] using Tabu Search for the compilation. The results of this comparison are presented in Table III. This comparison[1] was only possible for rectangular architectures, a limitation of [4]. As we can see from Table III, our LS-based compilation is able to obtain good quality results using a much shorter runtime (milliseconds vs 1 hour). Also, the average percentage deviation from the near-optimal result is 5.5%, hence, it can successfully be used for design space exploration. Next, we wanted to determine the increase in $\delta_{\mathcal{G}}$ computed by LSPR as the number of permanent faults k increases. Table IV shows the comparison between $\delta^k_{\mathcal{G}}$ for $k = 0$, 1 and 2. As an input to LSPR we have used an application-specific architecture, synthesized using our SA approach such that it minimizes the cost for each particular case-study and is tolerant to 2 faults. The cost of this architecture is presented in column 2 of Table IV, and the $\delta^k_{\mathcal{G}}$ results are in columns 3, 4 and 6, for k=0, 1 and 2, respectively. As we can see from Table IV, the increase in $\delta_{\mathcal{G}}$ is on average 11.8% for each increase in k.

The focus of the paper is to determine if application-specific architectures are more cost-effective than rectangular architectures. Thus, we have used our SA optimization approach

[1]As in [4], we ignored routing in this set of experiments.

TABLE V: Application-specific synthesis results obtained by SA

	$k=0$				$k=1$				$k=2$			
App.	Arch	Cost	C_{SA}	T_{SA}	Arch	Cost	C_{SA}	T_{SA}	Arch	Cost	C_{SA}	T_{SA}
PCR	7×10 (1,1,1)	79	60	14	7×10 (1,1,1)	79	65	38	9×11 (1,1,1)	108	98	50
IVD	7×10 (2,2,2)	88	62	16	7×10 (2,2,2)	88	70	58	10×8 (2,2,2)	98	85	45
CPA	7×8 (2,1,2)	71	59	10	7×8 (2,1,2)	71	66	20	11×12 (2,1,2)	147	127	30

from Section V to synthesize architectures for the case studies considered. The results are presented in Table V. Together with the results obtained by SA, we have also determined, using exhaustive search (which varies the $m\times n$ dimensions and the number of reservoirs), the cheapest general purpose architecture (column 3), which can run the application within the deadline. The size of the architectures for $k=0$, 1 and 2 are presented in columns 2, 6 and 10, respectively (the number in parentheses refer to the numbers of reservoirs for buffer, sample and reagent) and their cost is in columns 3, 7 and 11. The results of SA for $k=0$ are presented in columns 4 and 5 (C_{SA} is the architecture cost and T_{SA} is the runtime[2] of the SA algorithm). As we can see from Table V, our SA is able to produce application-specific architectures which are significantly cheaper than the best general purpose architecture.

Our synthesis methodology can also support the designer in performing a trade-off 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 12 for SA.

VII. Conclusions

We have proposed an SA-based synthesis approach for application-specific fault-tolerant DMB architecture, such that the architecture cost is minimized and the deadlines are satisfied even in the case of permanent faults. Every solution visited by SA is evaluated using a LS-based heuristics (LSPR), which performs a compilation of the application on the architecture. LSPR takes into account the worst-case overhead due to permanent faults and performs binding, scheduling, placement and routing evaluation.

As the experimental results show, our synthesis approach is able to significantly reduce the costs compared to general-purpose rectangular architectures. In addition, by synthesizing fault-tolerant architectures, our methodology can help the designer increase the yield of DMBs.

References

[1] K. Chakrabarty, R. B. Fair, and J. Zeng, "Design tools for digital microfluidic biochips: Toward functional diversification and more than Moore," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2010.

[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*, 2004.

[3] 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. on CAD of Integrated Circuits and Systems*, 2011.

[4] E. Maftei, P. Pop, and J. Madsen, "Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices," *Design Automation for Embedded Systems*, 2010.

[2]For the cooling schedule we used $TI=40$, $TL=100$ and $\varepsilon=0.97$.

[5] R. S. S. et all, "Digital microfluidic platform for multiplexing enzyme assays: Implications for lysosomal storage disease screening in newborns," *Clinical Chemistry*, 2011.

[6] F. Su and K. Bazargan, "Yield enhancement of reconfigurable microfluidics-based biochips using interstitial redundancy," *ACM JETC*, 2006.

[7] T. Xu and K. Chakrabarty, "Fault modeling and functional test methods for digital microfluidic biochips," *IEEE T-BCAS*, 2009.

[8] E. Maftei, P. Pop, and J. Madsen, "Droplet-aware module-based synthesis for fault-tolerant digital microfluidic biochips," *DTIP*, 2012.

[9] Y. Zhao, K. Chakrabarty, R. Sturmer, and V. Pamula, "Optimization techniques for the synchronization of concurrent fluidic operations in pin-costrained digital microfluidic biochip," *IEEE Trans. on VLSI Systems*, 2012.

[10] E. Maftei, P. Pop, and J. Madsen, "Routing-based synthesis of digital microfluidic biochips," *Design Automation for Embedded Systems*, 2012.

[11] ——, "Module-base synthesis of digital microfluidic biochips with droplet-aware operation execution," *ACM JECT*, in press.

[12] M. Alistar, P. Pop, and J. Madsen, "Online synthesis for error recovery in digital microfluidic biochips with operation variability," *DTIP*, 2012.

[13] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling and optimization of fault-tolerant distributed embedded systems with transparency/performance trade-offs," *ACM TECS*, 2006.

[14] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k," *Computer Science Technical Reports, Cornell University*, 1973.

[15] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test*, 2000.

[16] S. M. Sait and H. Youssef, *VLSI physical Design Automation: Theory and Practice*. Wspc, 1999.

[17] E. K. Burke and G. Kendall, *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2005.