Des Autom Embed Syst
DOI 10.1007/s10617-014-9157-2

Redundancy optimization for error recovery in digital microfluidic biochips

Mirela Alistar · Paul Pop · Jan Madsen

Received: 5 November 2013 / Accepted: 30 November 2014
© Springer Science+Business Media New York 2015

Abstract Microfluidic-based biochips are replacing the conventional biochemical analyzers, and are able to integrate all the necessary functions for biochemical analysis. The digital microfluidic biochips are based on the manipulation of liquids not as a continuous flow, but as discrete droplets. Researchers have proposed approaches for the synthesis of digital microfluidic biochips, which, starting from a biochemical application and a given biochip architecture, determine the allocation, resource binding, scheduling, placement and routing of the operations in the application. During the execution of a bioassay, operations could experience transient errors (e.g., erroneous droplet volumes), thus impacting negatively the correctness of the application. Researchers have proposed fault-tolerance approaches, which apply predetermined recovery actions at the moment when errors are detected. In this paper, we propose an online recovery strategy, which decides during the execution of the biochemical application the introduction of the redundancy required for fault-tolerance. We consider both time redundancy, i.e., re-executing erroneous operations, and space redundancy, i.e., creating redundant droplets for fault-tolerance. Error recovery is performed such that the number of transient errors tolerated is maximized and the timing constraints of the biochemical application are satisfied. The proposed redundancy optimization approach has been evaluated using several benchmarks.

Keywords Microfluidic biochips · Electrowetting on dielectric · Droplet-based biochips · Computer-aided design · Fault-tolerance · Runtime recovery

1 Introduction

Microfluidics, the study and handling of small volumes of fluids, is a well-established field, with over 10,000 papers published every year [1]. With the introduction at the beginning of 1990s of microfluidic components such as microvalves and micropumps, it was possible to

M. Alistar (✉) · P. Pop · J. Madsen
Technical University of Denmark, Lyngby, Denmark
e-mail: mali@dtu.dk

Published online: 13 January 2015

realize "micro total analysis systems" (μTAS), also called "lab-on-a-chip" and "biochips", for the automation, miniaturization and integration of complex biochemical protocols [2]. The trend today is towards *microfluidic platforms*, which according to [2], provide "a set of fluidic unit operations, which are designed for easy combination within a well-defined fabrication technology", and offer a "generic and consistent way for miniaturization, integration, customization and parallelization of (bio-)chemical processes". Microfluidic platforms are used in many application areas, such as, *in vitro* diagnostics (point-of-care, self-testing), drug discovery (high-throughput screening, hit characterization), biotech (process monitoring, process development), ecology (agriculture, environment, homeland security) [2–5].

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) [6]. 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 (discrete amount of fluid of nanoliters volume) [7].

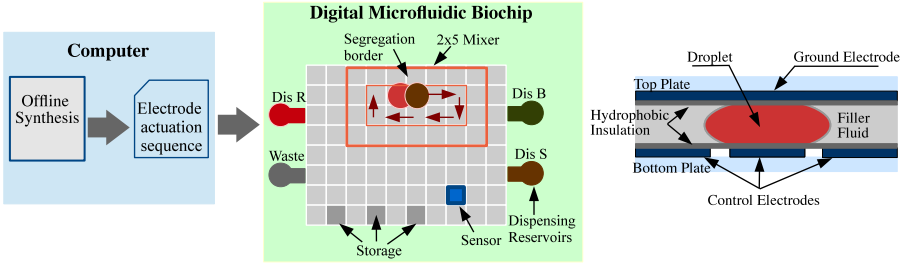
To be executed on a DMB, a biochemical application has to be synthesized. There is a significant amount of work on the synthesis of DMBs [5,8–12], which typically consists of the following tasks:

– *modeling* of the biochemical application functionality and biochip architecture;
– *allocation*, during which the needed modules are selected from a module library;
– *binding* the selected modules to the biochemical operations in the application;
– *placement*, during which the positions of the modules on the biochip are decided;
– *scheduling*, when the order of operations is determined and
– *routing* the droplets to the needed locations on the biochip.

The output of these synthesis tasks is the "electrode actuation sequence", applied by a control software to run the biochemical application. The control software executes on a computer connected to the biochip, as schematically represented in Fig. 1a.

Errors can occur during the execution of fluidic operations due to permanent faults (e.g., dielectric breakdown) or transient faults (e.g., unbalanced split due to unequal actuation voltages). The types of faults that happen in DMBs are discussed in [13]. Researchers have addressed permanent faults in the context of DMBs, for example by introducing a regular pattern of redundant electrodes [13,14].

In this paper we focus on transient faults. While a bioassay is executed on a DMB, the droplets will experience changes in volume during mixing, dilution and split operations.



(a) Biochip architecture model example **(b)** Electrowetting on dieletric (EWOD)

Fig. 1 Biochip architecture model example

Assuming ideal conditions, when two droplets merge for a mixing operation, the resulting droplet has a volume equal with the sum of the input droplets volumes. After a split operation, the resulting droplets have volumes equal to half of the initial droplet volume. However, the volume of a droplet can also vary erroneously due to transient faults, such as an electrode coating fault or unequal actuation voltages during split [13]. The erroneous droplet volume propagates throughout the execution of the bioassay, thus impacting negatively the correctness of the application. Bioassays have high accuracy requirements, determined by the acceptance range for the volume and concentration of droplets. Example applications with accuracy requirements of less than ± 10 % are drug discovery applications [15] and plasmid DNA preparation [16]. Hence, it is imperative to introduce fault-tolerance to transient errors.

1.1 Related work

Past research has addressed the erroneous volume variation due to transient faults. Thus, in [17], we have focused on the erroneous volume variation after an unbalanced split operation. If an error is detected during a split operation the resulted droplets are merged back. The merging operation is instantaneous. Our proposed scheduling algorithm derives *offline* (i.e., at design time) the schedules needed to recover from all combinations of faulty split operations. *Online* (i.e., during the runtime of the biochemical application), the scheduler will switch to the backup schedules containing the electrode actuation sequence to tolerate the observed error occurrences.

The work in [18] addresses the volume variations in all types of operations, not only split operations. Thus, intermediate droplets of correct volumes are stored at checkpoints. When an error is detected, the stored droplets are used in the recovery subroutine. The locations of the checkpoints and the recovery subroutines are determined offline and stored in a microcontroller memory. If an error is detected during runtime at a checkpoint, the microcontroller interrupts the bioassay, and transports the intermediate product droplets to the storage units; then the corresponding recovery subroutine is executed using a statically predetermined allocation and placement, which do not consider the current context. A method to precompute and store a dictionary that contains recovery solutions for all combinations of errors is proposed in [19]. When an error is detected, the system looks in the dictionary for the corresponding recovery actuation sequence. Compression algorithms are used to reduce the size of the dictionary in order to store it on the flash memory of the microcontroller. In all mentioned approaches, the error recovery actions are determined *offline*, and are applied online when an error is detected.

Researchers have also proposed *online* approaches that determine the necessary recovery actions *during the execution of the biochemical application*, at the moment when an error is detected. Such online recovery approaches, some of which also perform online re-synthesis to reconfigure the electrode actuation sequence, are possible because the biochemical application execution times are much slower compared to the control software execution. The work in [20] addresses sample preparation and proposes dynamic error recovery to recreate online the desired target concentrations, using the stored intermediate droplets. A general approach, that synthesizes a new implementation containing the appropriate error recovery actions whenever errors are detected, is proposed in [21]. The online synthesis re-computes the placement of operations and the droplets routes using a List-Scheduling based implementation. In [22], we have proposed an online error recovery approach, which re-synthesizes at runtime the application to tolerate transient errors in all types of operations.

1.2 Contributions

Fault-tolerance is achieved through redundancy. Time redundancy (re-execution of operations in case an error is detected) uses the available slack in the schedule to recover from failure. Sometimes, there is no available slack and hence time redundancy can lead to a deadline miss. Space redundancy (creating redundant droplets) makes use of available biochip area to execute replicas of operations in order to recover from failure.

Previous work has used a single type of recovery technique (e.g., re-execution [17], checkpointing [18]) and has not used space redundancy for fault-tolerance. The only exception is our work in [22] where we have considered a combination of time and space redundancy techniques. In all the previous work, including [22], the assignment of recovery techniques (re-execution, checkpointing, space redundancy) to operations has been *manually* decided at design time (*offline*). In this paper, we propose an online recovery strategy, which decides *online* and *dynamically* the optimization of redundancy required for fault-tolerance. As shown in the experimental results, our proposed strategy is able to obtain better quality results, in terms of application completion time, compared to the solutions in the literature. The faults we address in this paper are transient, i.e., they can occur at any time during the execution of the application. Hence, having a strategy for a fast recovery is beneficial, as more transient faults can be tolerated within the deadline.

The contributions of this paper are the following:

(1) In Sect. 6, we propose an online redundancy optimization approach, which decides at runtime, depending on the unpredictable error occurrences, the assignment of the appropriate redundancy techniques (time or space redundancy) for the operations of the biochemical application. Our online optimization approach decides between time redundancy (re-execution of operations) and space redundancy (creating redundant droplets) depending on the current error scenario. In this paper, time redundancy re-executes the faulty operations to recover *after* an error is detected. Space redundancy uses the available biochip area to execute replicas of operations *before* we know an operation is erroneous. If enough area is available, space redundancy will not lead to time delays. The redundancy optimization is performed such that the number of transient faults tolerated is maximized and the timing constraints of the biochemical application are satisfied.

(2) In Sect. 3.3, we propose a generalized fault-tolerant application model, which captures the extra operations needed for redundancy. Our proposed application model considers both space and time redundancy techniques and can tolerate any number of transient faults.

(3) In Sect. 6.3, we propose an algorithm for generating recovery subgraphs which contain the required redundant operations needed for fault-tolerance. The algorithm takes advantage of the existing redundant droplets such that the recovery time is minimized. Example redundant droplets are by-product droplets intended for discarding (e.g., produced by a dilution operation) or unused droplets that have been speculatively generated by space redundancy.

This paper is organized in nine sections. Sections 2 and 3 present the architecture and application models, respectively. In Sect. 3.2, we discuss the redundancy techniques used for fault-tolerance. We formulate the problem in Sect. 4 and use an example to illustrate it. We present our approach to online redundancy optimization and recovery for the case when a sensor is used for detection (Sects. 5 and 6) and when a charged-couple device (CCD) camera-based detection system is used (Sect. 7). The proposed algorithms are evaluated in Sect. 8, and Sect. 9 presents our conclusions.

Table 1 Module Library [32]

Operation	Area	Time (s)
Mix	2×5	2
Mix	2×4	3
Mix	3×3	7
Mix	1×3	5
Mix	2×2	10
Dilute	2×5	4
Dilute	2×4	5
Dilute	3×3	10
Dilute	1×3	7
Dilute	2×2	12
Store	1×1	N/A
Transport	1×1	0.01

2 Biochip architecture model

In a DMB, a droplet is sandwiched between a top ground-electrode and bottom control-electrodes, see Fig. 1b. The droplet is separated from electrodes by insulating layers and it can be surrounded by a filler fluid (such as silicone oil) or by air. 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 [6]. For example, in Fig. 1b, if the control-electrode on which the droplet is resting is turned off, and the left control-electrode is activated by applying voltage, the droplet will move to the left. A biochip is typically connected to a computer (or microcontroller) as shown in Fig. 1a 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, to run a biochemical application.

A DMB is modeled as a two-dimensional array of identical control-electrodes, see Fig. 1a, where each electrode can hold a droplet. There are two types of operations: reconfigurable (mixing, split, dilution, merge, transport), which can be executed on any electrode on the biochip, and non-reconfigurable (dispensing, detection), which are bound to a specific device such as a reservoir, a detector or a sensor. A mixing operation is executed when two droplets are moved to the same location and then transported together according to a specific pattern (see Fig. 1a). Considering the biochip in Fig. 1a, a droplet can only move up, down, left or right with EWOD, and cannot move diagonally. A split operation is done by keeping the electrode on which the droplet is resting turned off, while applying concurrently the same voltage on two opposite neighboring electrodes. For example, in Fig. 1b, to split the droplet, we have to turn off the control-electrode in the middle and turn on simultaneously the left and right control-electrodes. Dilution is a mixing operation followed by a split operation. Each reconfigurable operation is executed in a determined biochip area, called a "module". For example, the two droplets from Fig. 1a are mixing on a 2×5 module, by moving according to the indicated pattern. Based on experiments, researchers characterize a module library $\mathcal{L}$, such as the one in Table 1, which provides the area and corresponding execution time that are needed for each operation. As shown in Table 1, the time needed for two droplets to mix on a 2×5 module is 2 s. In case two droplets are on neighboring electrodes, they merge instantly. To avoid accidental merging, each module is surrounded by a "segregation border " of one-electrode thickness, see Fig. 1a.

The biochip contains non-reconfigurable devices such as input (dispensing) and waste reservoirs, sensors and actuators, on which the non-reconfigurable operations are performed. For example, the biochip from Fig. 1a has four dispensing reservoirs, one for buffer, one for sample, one for reagent and one waste reservoir. 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 afterwards split to obtain the final droplet [23]. Sensors can be used to determine the result of the bioassay or for error detection. For example, a LED and a photodiode combination is used as detector for glucose concentration in a droplet [2,24]. The location of these non-reconfigurable devices is fixed on the biochip array. Fig. 1a shows the location of reservoirs and a sensor, which are placed on a biochip architecture of 10×8 electrodes. Example actuators are heaters [1] and filters [2].

In this paper we are interested in the use of sensors for error detection. A LED and photodiode sensor can be used for determining the concentration of a specific compound in a droplet (e.g., glucose [4,24]), whereas a capacitive sensor [23] can be used to determine the volume of a droplet. The capacitive-sensing circuit used to measure the volume operates at high frequency (15 KHz [6]), while the LED-photodiode sensor needs 5 seconds to measure the absorbance of the product droplet in order to determine its concentration. For the photodiode detector, a transparent droplet has to be mixed with a reagent to generate a colored analyte. In this case, the initial droplet is not suitable for other operations. The capacitive sensor does not alter the initial droplet, which can be used for subsequent operations.

Erroneous droplet volumes can also be detected by using a CCD camera-based detection system, which analyzes the images captured during the bioassay execution [25]. The CCD camera-based detection system adds to the complexity of the system by requiring external instruments and specialized software, but has the advantage of detecting the errors when they occur, eliminating the need for specialized detection operations, which have to transport a droplet to a sensor on the biochip. In this paper we are interested to tolerate transient faults, which result in erroneous droplet volumes. We consider both capacitive sensors and a CCD camera-detection system for determining the volume of a droplet, which is then compared to its expected volume in order to perform error detection.

2.1 Fault models

Many biochemical applications, such as drug development and clinical diagnostics, have high accuracy requirements. DMBs can be affected by faults, resulting in failure to complete the application or in an incorrect result of the bioassay. Hence, researchers have addressed faults by proposing fault models [13], testing and detection methods [26,27] and error recovery strategies [25,28]. Faults can be classified in two main categories: permanent faults and transient faults.

Permanent faults Also known as "catastrophic faults", permanent faults are caused by defects, such as dielectric breakdown and degradation of the insulator, introduced usually during the fabrication of the DMBs. Permanent faults prevent the operation from executing. Detailed information about permanent faults can be found in [27,28]. Researchers have proposed several methods of testing for permanent faults in DMBs [29,30].

Transient faults Also known as "parametric faults", transient faults occur unpredictably during the execution of an operation. Although transient faults do not prevent the operation from executing, the result of the operation does not correspond to its specified behavior. For example, the misalignment between the droplet and the control-electrode is the most

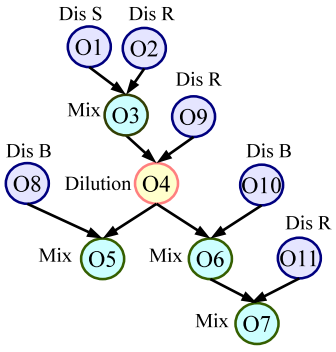
frequent cause of unbalanced split operation. Erroneous variation in droplet volume can have a significant negative impact on the outcome of the biochemical application. Estimates show that erroneous variation in droplet volume can count to up 80% of the total error in a bioassay [31]. As mentioned, DMBs can have integrated sensors that operate at a speed comparable to the execution time of a fluidic operation. Such sensors facilitate real-time error detection and recovery from transient errors.

In this paper, we focus on transient faults during operation execution. We consider that an operation is faulty if the volume of the outputted droplet exceeds the acceptable boundaries given by the accuracy requirements of the application. We propose a strategy that decides online the necessary recovery actions to perform error recovery.

3 Biochemical application model

A biochemical application is modeled using an acyclic directed graph [3], where the nodes represent the operations, and the edges represent the dependencies between them. We have extended the model proposed in [3] to model error detection and error recovery and to capture the operations needed for time and space redundancy.

We denote with $\mathcal{G}^0$ the biochemical application model without fault-tolerance features. Fig. 2a presents such an application graph $\mathcal{G}^0$ with 11 operations. A node in $\mathcal{G}^0$ represents an operation O_i. In Fig. 2a we have operations O_1 to O_{11}. A directed edge e_{ij} between operations O_i and O_j models a dependency: O_j can start to execute only when it has received the input droplet from O_i. An operation is ready to execute only after it has received all its input



(a) Application graph $\mathcal{G}^0$

i) $\epsilon_{Ds} = E_{Ds}$

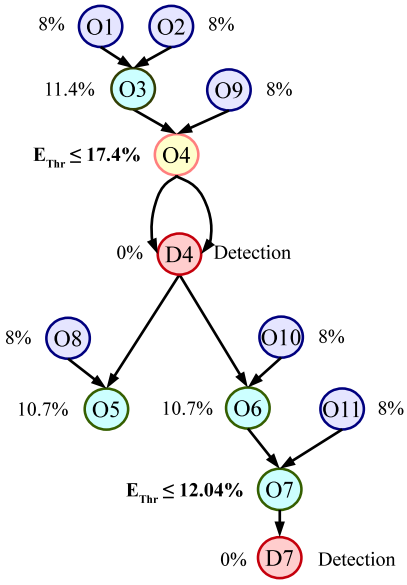
ii) $\epsilon_{Trans} = \sqrt{I^2 + E_{Trans}^2}$

iii) $\epsilon_{Slt} = \sqrt{I^2 + (2E_{Slt})^2}$

iv) $\epsilon_{Mix} = \sqrt{(0.5I_1)^2 + (0.5I_2)^2 + E_{Mix}^2}$

v) $\epsilon_{Dlt} = \sqrt{(0.5I_1)^2 + (0.5I_2)^2 + (2E_{Dlt})^2}$

(c) Error propagation equations [9]



(b) Application graph $\mathcal{G}$ with detection operations

Fig. 2 Example application model, with error propagation and detection

droplets. For example, a mixing operation O_7 is ready to execute only after operations O_6 and O_{11} have finished executing and the droplets have been transported to the biochip area where O_7 will perform the mixing. If the produced droplet cannot be used immediately (e.g., has to wait for another operation to finish), it has to be stored in a storage unit (see Table 1) to avoid accidental merging. In our model, we do not capture explicitly the routing operations required to transport the droplets, but we take routing into account during the synthesis. We use the data from [6], thus we assume that routing a droplet between two adjacent electrodes takes 0.01 s (see the "Transport " operation in Table 1). A droplet is dispensed in 2 s [32]. For simplicity reasons, we have ignored routing in all the examples in this paper, but we take routing into account during synthesis.

Biochemical applications can have strict timing constraints. For example, in the case of sample preparation, the reagents degenerate fast, affecting the efficiency of the entire bioassay [20]. Researchers have so far used a hard deadline $d_{\mathcal{G}}$ associated to an application graph $\mathcal{G}$ to capture such constraints. For safety-critical applications, this is a safe conservative assumption. However, other applications may have a soft deadline, since there is still some utility in continuity to execute the application also after a deadline. Operations can also experience execution time variability due to the variability of some biochemical processes [33]. In addition, operations can have local deadlines. For example, once two droplets are mixed, they should not wait more than a certain time before they are subsequently used (e.g., the reactions of aryllithium compounds bearing alkoxycarbonyl groups [34]). We can easily model such local deadlines by introducing dummy nodes in the application graph, and by having a global application deadline. In this paper we assume a hard deadline $d_{\mathcal{G}}$, but our approach can be extended to handle soft deadlines, with the aim of maximizing the utility. For example, our List Scheduling-based online synthesis heuristics can be extended as in [35] to handle both hard and soft deadlines.

3.1 Fault propagation and error detection

In this paper, we define a *fault* as an unintended volume variation of a droplet from a nominal volume. Not all faults are errors. Thus, volume variations (faults) may occur repeatedly and they may propagate through the execution of the application, but they are not considered errors. We assume that for every application, according to its specific accuracy requirements, the designer decides on a specific volume variation boundary E_{Thr}, named error threshold, which is the maximum permitted variation from the nominal volume. We define an *error* as a volume variation that exceeds the error threshold E_{Thr}. In this section we discuss how we determine where to introduce detection operations in order to detect the errors. Our proposed fault-tolerance approach will recover from errors by recreating the affected droplets. We define as *failure* the situation when the recovery has introduced delays such that the application deadline is no longer satisfied.

In [18], the authors use fault analysis [36] to derive the fault limit at the output of an operation from its intrinsic fault limit and the limits of the input operations. Each fluidic operation has a specific variation range associated with it, called "intrinsic fault limit", which captures the worst-case volume variations. For example, if the intrinsic fault limit E_{Mix} for mixing is 10 %, after a mix operation the output droplet can have a volume between 90 and 110 % of the nominal value. We use the following notation: E_{Mix} is the intrinsic fault limit for mixing operation, E_{Dlt} for dilution, E_{Trans} for transport, E_{Ds} for dispensing, E_{Slt} for split. Experimentally, the following values were determined for the intrinsic fault limits: $E_{Ds} = E_{Dlt} = E_{Slt} = 8$ %, $E_{Mix} = 10$ %, $E_{Trans} = 12$ % [18]. The equations in Fig. 2c [18] calculate the fault limit ϵ_{Mix} at the output of a mixing operation, ϵ_{Ds} for

dispensing, ϵ_{Dlt} for dilution, ϵ_{Trans} for transporting and ϵ_{Slt} for split operations as a function of intrinsic fault limits E_{Mix}, E_{Ds}, E_{Dlt}, E_{Trans} and E_{Slt} respectively, and input fault limits I_1 and I_2. The fault limit at the output of an operation is propagated and becomes the fault limit for its successor operation. In Fig. 2b, for the dilution operation O_4 we have the intrinsic fault limit $E_{Dlt} = 8\,\%$ and the input operation fault limits $I_1 = 11.4\,\%$ (for O_3) and $I_2 = 8\,\%$ (for O_9). Using Eq.(v) from Fig. 2c, we estimate the fault limit at the output of O_4 to be 17.4 %.

We continue to calculate the fault limits for all fluidic operations in the biochemical application. When the fault limit after an operation O_i, calculated according to the presented error analysis, exceeds the error threshold E_{Thr}, a detection operation D_i is inserted into $\mathcal{G}^0$ to detect at runtime if an error has actually occurred or not. For the graph in Fig. 2a, the E_{Thr} was set to 12 %; as a result, the detection operations D_4 and D_7 were inserted into $\mathcal{G}^0$ after O_4 and O_7, respectively, obtaining $\mathcal{G}^+$, as depicted in Fig. 2b. In case O_i is an operation with two output droplets (e.g. the dilution operation O_4 in Fig. 2b), the detection operation will have two inputs, as in the case with operation D_4 in Fig. 2b. However, it is sufficient to measure the volume of only one droplet in order to determine if an error has occurred.

We will initially assume that the volume is measured using a capacitive sensor, which means that the droplet has to be transported to the sensor. This is what we assume will happen in a detection operation D_i. However, in Sect. 7 we will discuss the alternative of using a CCD camera-based sensor. Such a setup could determine the droplet sizes during a detection operation D_i, or it could track the droplets continuously to determine erroneous volumes, without the need to insert detection operations as discussed in this section.

Two outcomes are possible after a detection operation. The first one corresponds to a correct droplet volume, and the second one to an erroneous droplet volume. In case the measured volume is the expected one, i.e. no error has occurred, the corresponding droplet is transported from the sensor to the location where the subsequent operations will execute. Otherwise, if the measured volume is outside the expected boundaries, the available recovery mechanism is triggered, and the volume of the droplet is brought back to the nominal value. Thus, after each detection, we reset the fault limit to 0 %, since it is assumed that in case an error is detected, the necessary actions to recover from the error are taken. The assumption is that a volume error occurring in an earlier operation can also be detected later, after it has propagated. For operations where this is not the case, the designer will statically assign a corresponding detection operation at a pre-determined place in the graph. Researchers have so far assumed that all the detection operations are statically assigned. However, our online redundancy optimization and recover strategy will assign dynamically the detection operations by adjusting at runtime the error threshold E_{Thr} based on the current error occurrences, see Sect. 6.1.

3.2 Error recovery

If the volume of a droplet is detected as erroneous, we have to create a new similar droplet with the correct volume (i.e., we recover from the detected error). This can be done in several ways. The simplest solution is to discard all the operations executed so far and re-execute the entire application from the beginning. However, this is very time-consuming, especially for the cases when errors occur at later stages. For most applications, a complete re-execution results in exceeding the deadline and wasting expensive reagents and hard-to-obtain samples. For example, in Fig. 2b, if an error is detected in D_7, we have to regenerate the droplets needed for O_7. In this case, we do not need to re-execute operations O_5 and O_8.

In our approach, we use three strategies to create droplets with the correct volume:

(1) We re-execute the operations needed to re-generate the droplet, *after* an error has been detected. We call such an approach *time redundancy*. The advantage of time redundancy is that it re-executes operations only when needed (when an error has been detected); the disadvantage is that it leads to delays in application execution.
(2) We execute operations which will produce a correct droplet *before* we know if an error has occurred, in parallel to the application execution. We call this approach *space redundancy*. The advantage of space redundancy is that, if an error is detected, we can use the redundant correct droplet directly, without waiting to be re-generated. The goal is to use the extra biochip area, if available, to speculatively produce correct droplets, without a negative impact on the application execution. The disadvantage is that if not enough area is available, space redundancy will introduce delays during the application execution, since it competes for the same resources with the regular operations.
(3) We use the redundant droplets available as a by-product of the regular application execution or after using space and time redundancy for other operations. For example, if we use only one droplet after a dilution operation, we can use the second droplet for fault-tolerance, if it has the correct volume. Let us assume that we need to re-generate the droplets for an operation O_j. If we predicted an error in a predecessor operation O_i of O_j, and we used space redundancy for O_i but an error has not been detected after O_i, we may be able to use in O_j some of the redundant droplets produced by space redundancy for O_i.

Our online error recovery strategy will decide at runtime which redundancy technique to use such that the number of transient faults tolerated is maximized and the application deadline is satisfied. We also take advantage of any redundant droplets available at runtime to reduce the recovery times. These aspects are discussed in Sect. 6.3.

In all three strategies outlined earlier, we generate the correct droplets by using redundant operations in the application graph, corresponding to a detection operation D_i. These redundant operations are grouped into a subgraph R_i, which is connected to the graph $\mathcal{G}^+$, i.e., the application graph $\mathcal{G}^0$ with the detection operations. These subgraphs are responsible for producing correctly-sized droplets, and are inserted into $\mathcal{G}^+$ such that output droplets produced by R_i become the input droplets for the successors of operation D_i. Fig. 3b shows the recovery subgraph R_{11} for detection D_{11} in the graph in Fig. 3a. A recovery subgraph R_i can be obtained at design time by performing a breadth-first search on the graph $\mathcal{G}^+$, starting from O_i and going backwards towards the inputs. Note that not all the operations in R_i will be needed at runtime because redundant droplets may be already available, as discussed at point 3 earlier. Our online strategy will carefully manage these redundant droplets and will eliminate from R_i, at runtime, the superfluous operations for which such droplets are available, see the algorithm in Fig. 4.

3.3 Generalized fault-tolerant application model

Let us now discuss the difference between time and space redundancy in terms of how the subgraph R_i is connected to the application graph $\mathcal{G}^+$ and how it is executed in case of time and space redundancy. A *conditional edge* is a dependency between two operations, which is activated only when the associated condition is true. Conditional edges are used to model the outcome of a detection operation D_i. Let us assume that D_i will produce an error condition E_i, which is true if an error has been detected and false if an error has not been detected. Thus, D_i will have two outgoing conditional edges, labeled with E_i and $\overline{E}_i$. We call such

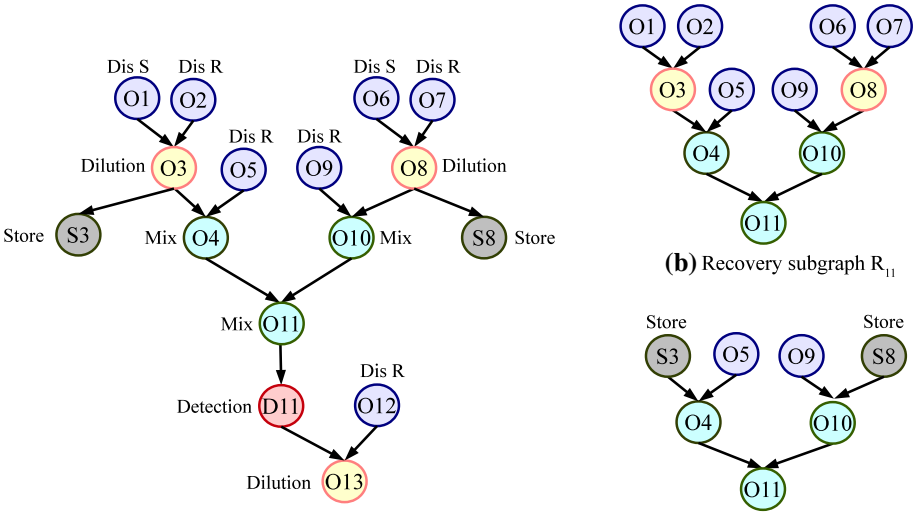


Fig. 3 Example of recovery subgraph

an operation with outgoing conditional edges a *disjunction node*. An *execution guard* is a condition which has to be true in order to activate the operations of a redundant subgraph R_i.

Time redundancy Fig. 5a presents how the subgraph R_i is connected to the graph $\mathcal{G}^+$ in case of time redundancy for an operation O_i followed by a detection D_i. The subgraph R_i is depicted using a rectangular node. Such a node is hierarchical, since it contains all the operations of R_i. Note that an error can occur also during the execution of the subgraph R_i used for recovery. We denote with D_i^R the detection operation needed to detect such an error, which occurs during the recovery. We denote with E_i and E_i^R the error conditions produced after the detection operations D_i and D_i^R, respectively.

With time redundancy, the subgraph R_i is activated if an error is detected by D_i or by D_i^R, i.e., if $E_i \vee E_i^R$ is true. This is depicted in Fig. 5a with an arrow on top of the rectangular node R_i, labeled with the execution guard $E_i \vee E_i^R$. Let us denote with O_B the successor operation of O_i (corresponding to the detection D_i). O_B will be activated only if no error is detected by D_i or no error is detected by D_i^R after the recovery subgraph R_i. This is captured in our model by connecting O_B with the conditional edges $\overline{E}_i$ and $\overline{E}_i^R$ to D_i and D_i^R, respectively. If an error is detected by D_i or D_i^R, the corresponding incorrectly sized droplets will have to be discarded. This is achieved by inserting the operations O_A and O_C in the graph and connecting them to D_i and D_i^R using the conditional edges E_i and E_i^R, respectively. The operations O_A and O_C are responsible to transport the incorrect droplets to the waste reservoirs. In these cases, i.e., E_i or E_i^R are true, R_i is activated, as discussed. Section 6 presents how R_i is synthesized, including how its operations are scheduled, in order to be executed. We also synthesize the operations O_A and O_C which transport the incorrect droplets to the waste.

Because D_i^R detects an error during R_i and thus activates it again for execution, our time redundancy model tolerates several transient faults, constrained only by the deadline $d_{\mathcal{G}}$.

Space redundancy Fig. 5b presents our space redundancy model. We use space redundancy to tolerate a *single* transient error detected during a detection operation D_i. If a second

Fig. 4 Algorithm for determining the recovery subgraph

DetermineRecoverySubgraph($D_i, \mathcal{G}$)

1: L_{stg} - the list of stored droplets
2: Q - the list of recovery operations
3: R_i = BFS($D_i, \mathcal{G}$)
4: **for** each operation O_i in R_i **do**
5: **if** O_i has no successors **then**
6: $Q.push(O_i)$
7: **end if**
8: **end for**
9: **repeat**
10: $O_i = Q.pop()$
11: label O_i as explored
12: n = FindStoredDroplet(L_{stg}, O_i)
13: **if** $n \neq \emptyset$ **then**
14: PruneGraph(R_i, O_i)
15: Remove(L_{stg}, n)
16: **for** each predecessor O_j of O_i **do**
17: **if** O_j is not explored **then**
18: $Q.push(O_j)$
19: **end if**
20: **end for**
21: **end if**
22: **until** $Q = \emptyset$
23: **return** R_i

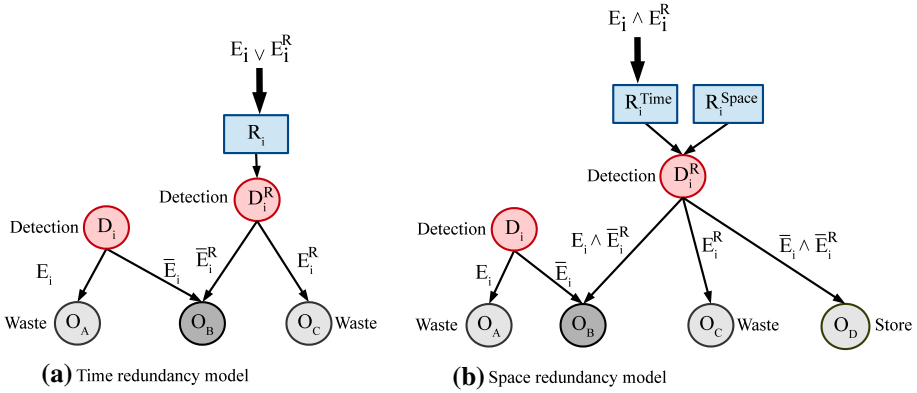


Fig. 5 Recovery using time vs. space redundancy

transient error is detected in the same place, we revert to time redundancy. We denote with R_i^{Space} the subgraph R_i used for space redundancy in Fig. 5b and with R_i^{Time} the one used for time redundancy. For a detection operations D_i, we do not introduce more than one subgraph for space redundancy because they consume biochip area and, if an error does not occur, too much space will be wasted.

Similar to time redundancy, we denote with D_i^R the operation needed to detect an error in R_i^{Space} or R_i^{Time}, with O_B the successor operation of O_i and with O_A and O_C we denote waste operations. The main difference to the time redundancy model from Fig. 5a is that

the subgraph R_i^{Space} used for space redundancy does not have an execution guard, i.e., it is executed regardless if an error is detected by D_i or not.

Section 6 discusses how our online strategy will synthesize R_i^{Space} and how it will schedule and execute it at the same time with the regular operations, on the available extra biochip area. If no error is detected during D_i, our online synthesis will stop the operations in R_i^{Space} which have not finished executing, if such operations exist, and will manage the resulted redundant droplets to be potentially used in future recoveries. This is not captured in our model from Fig. 5b.

The advantage of space redundancy is that if an error is detected by D_i, we do not have to wait for the re-execution of R_i to get the correct droplets, as it is the case with time redundancy. Instead, O_B is ready to execute using the redundant droplet produced by R_i^{Space}. This is captured in the model in Fig. 5b by the conditional edge $E_i \wedge \overline{E}_i^R$ from D_i^R to O_B, which is activated only if an error has occurred in D_i and no error has occurred during the execution of R_i^{Space}. That is, we only use the redundant droplet from R_i^{Space} if it is of correct volume, condition checked by D_i^R, to which R_i^{Space} is connected, and captured by $\overline{E}_i^R$. Note that O_B may have to wait for R_i^{Space} to finish executing, if not all operations in it have completed.

Our online recovery decides at runtime to use space redundancy only if enough extra resources are available, and will attempt to synthesize and schedule R_i^{Space} such that O_B does not have to wait. In case an error has been detected by D_i and R_i^{Space} has also experienced an error, which was detected by D_i^R, we will use time redundancy (R_i^{Time}) to recover from these two errors. Hence, R_i^{Time} is only activated if both E_i and E_i^R are true. Any errors in R_i^{Time} will be handled as discussed for time redundancy. Finally, if there are no errors at all in Fig. 5b (i.e., $\overline{E}_i \wedge \overline{E}_i^R$), we are left with redundant droplets produced by R_i^{Space}. Our online recovery will decide what to do with these droplets. For example, they can be stored to be used later during other recoveries. This is depicted in Fig. 5b with the "store" operation O_D, connected with the conditional edge $\overline{E}_i \wedge \overline{E}_i^R$ to D_i^R.

4 Problem formulation

In this paper we address the following problem. As input we have a biochemical application modeled as a graph $\mathcal{G}^0$ with a deadline $d_{\mathcal{G}}$, which is executed on a DMB modeled as a $m \times n$ array $\mathcal{A}$ of cells. A characterized library $\mathcal{L}$, containing the area and execution time for each operation (similar to Table 1), is also given as input. We are interested to determine online the necessary recovery actions, so that the number of transient faults tolerated is maximized and the application deadline $d_{\mathcal{G}}$ is satisfied.

In this paper we consider both time redundancy and space redundancy when deciding what fault-tolerant policy to use for each detection operation. We decide online where to introduce detection operations and which redundancy technique to use.

4.1 Motivational example

The advantage of using space redundancy is faster recovery time in case of error, at the cost of extra overhead in completion time, in case of no error. When time redundancy is used, the recovery actions are executed only after an error is detected, so no extra time overhead is added

in case of no error. However, in case of error, the recovery is slower for time redundancy than space redundancy. Past research has used time redundancy as error recovery in both offline [19] and online [21] strategies. We have proposed space redundancy as a recovery method in [22]; however the corresponding redundancy was assigned manually. Since the error scenarios are not known in advance, an online redundancy optimization strategy can better exploit the current configuration, leading to improved results.

Let us illustrate this by using the application graph $\mathcal{G}^0$ from Fig. 6a, which has a deadline $d_{\mathcal{G}} = 25$ s and has to be executed on the 10×8 biochip from Fig. 6c. In Fig. 7a we show the schedule of the application for the case when we do not consider the issue of fault-tolerance (and there are no errors). 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. As shown in Fig. 7a, operation O_1 starts executing at $t = 0$ s and finishes at $t = 2$ s. The completion time of $\mathcal{G}^0$ is $\delta_{\mathcal{G}^0} = 18$ s. Such a schedule has a one-to-one correspondence to the electrode actuation sequence, used by the control software on the computer that runs the biochemical application on the biochip, see Fig. 1a. As mentioned in the introduction, an implementation consists of allocation, binding, placement, scheduling and routing. The allocation and binding of operations to devices are shown in the Gantt chart as labels at the beginning of each row of operations. For example, the non-reconfigurable operation O_1 is bound to the dispensing reservoir $Dis\ S$, while the mixing operation O_3 is bound to $Mixer_1$, for which we have allocated a 2×5 module. The placement of modules, for all the examples in this section, is presented in Fig. 6c.

In this example, we are interested to tolerate two transient errors, affecting the volume of droplet. Two detection operations D_6 and D_8 are inserted in $\mathcal{G}^0$, obtaining the graph $\mathcal{G}^+$ from Fig. 6b. For the considered example, we have four possible error scenarios in the case

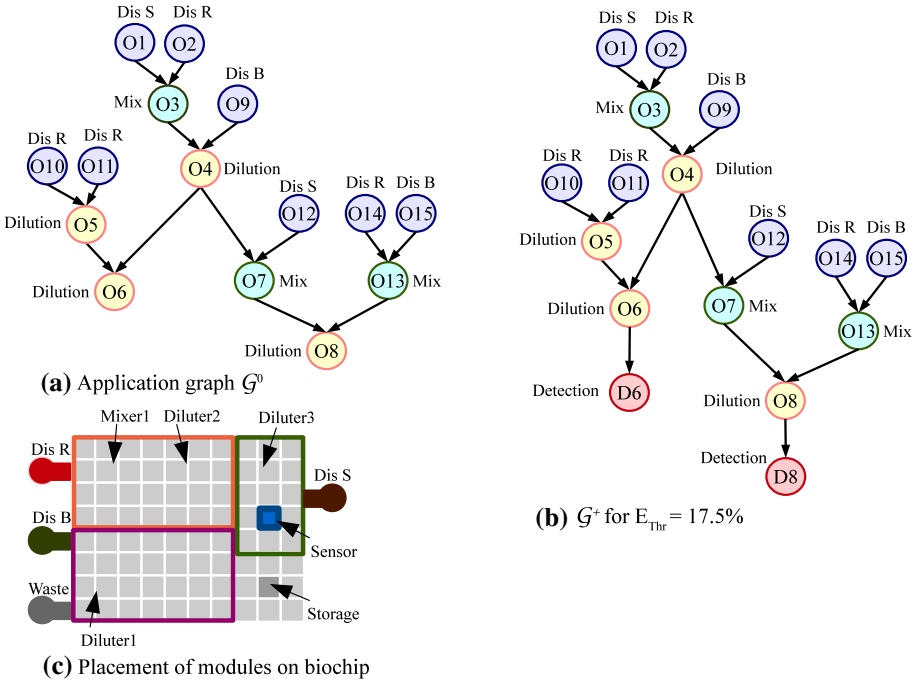


Fig. 6 Motivational example

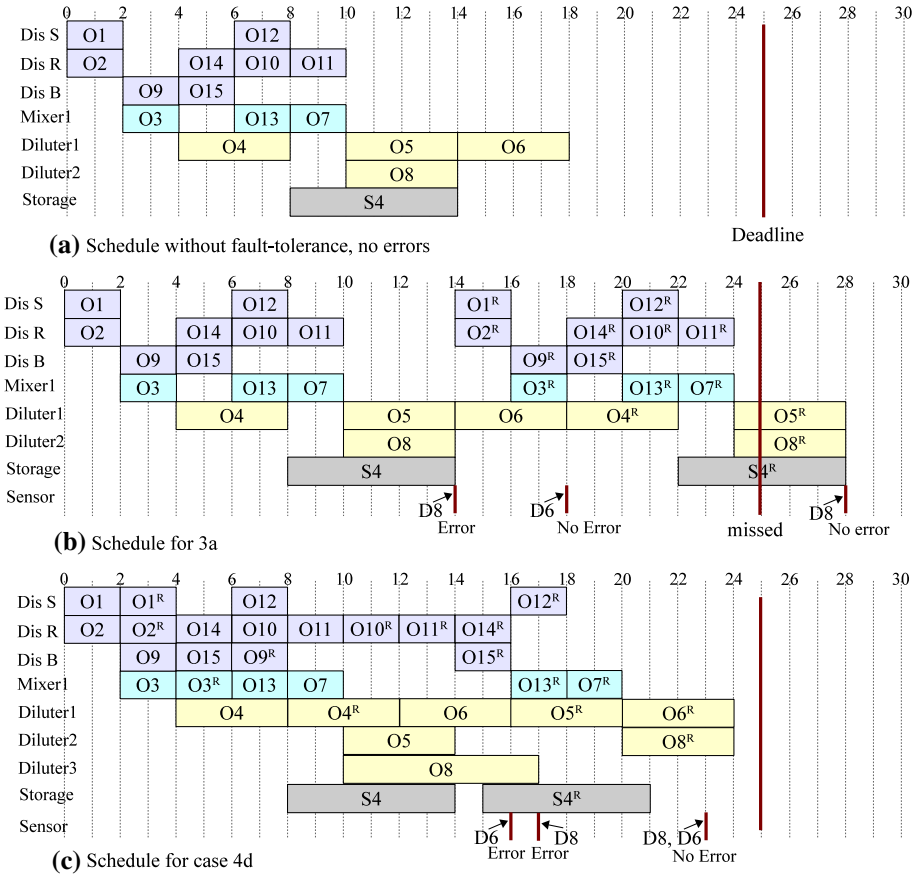


Fig. 7 Schedules for various error scenarios

of maximum two transient faults: (1) when no error is detected, (2) when a single transient error is detected by D_6, (3) when a single transient error is detected by D_8 and (4) when two transient errors are detected by both D_6 and D_8. These error scenarios are presented in the rows of Table 2. For this example we assume no errors during recovery. However, our approach also takes into account errors during the recovery operations.

There are several possible redundancy solutions to tolerate the transient faults in each scenario. We are interested to decide on an assignment of redundancy to the application, such that the deadline of 25 s is satisfied in every fault scenario (the application is fault-tolerant only if it completes within its deadline in all the possible fault scenarios). There are four possible solutions for our example: (a) using only time redundancy, (b) using only space redundancy, (c) using time redundancy for tolerating the error detected by D_6 and space redundancy for tolerating the error detected by D_8 and (d) using time redundancy for D_8 and space redundancy for D_6. The time and space redundancy subgraphs are added to $\mathcal{G}^+$ in Fig. 6b as discussed in Sect. 3.2. Columns 3 to 6 in Table 2 present the best results in terms of the application completion time $\delta_{\mathcal{G}}$ obtained using each redundancy scheme (a)–(d) for each error scenario (1)–(4). The completion times $\delta_{\mathcal{G}}$ that miss the deadline of 25 s are showed in parenthesis. In these situations, we consider that the application was not able to tolerate the transient errors.

Table 2 Application completion times [for combinations of error scenarios and redundancy scenarios]

Scenario	Detected by	Fault-tolerance solutions			
		(a) Only time redundancy	(b) Only space redundancy	(c) Time in D_6 Space in D_8	(d) Time in D_8 Space in D_6
1	–	18	18	18	18
2	$\{D_6\}$	(36)	(26)	(30)	22
3	$\{D_8\}$	(28)	24	23	23
4	$\{D_6, D_8\}$	(46)	(36)	(30)	24

As we see from Table 2, the only situation when the application is able to recover in all error scenarios and complete before the deadline is solution (d) when time redundancy is used in D_8 and space redundancy is used in D_6. The schedule length is 25 s, satisfying the deadline. In Fig. 7c we show the schedule for two errors, one in D_6 and one in D_8, error scenario (4), in case (d). If we use only time redundancy, as in case (a), we miss the deadline in error scenarios (2)–(4). The schedule for the case (a) for error scenario (3) is presented in Fig. 7b, and has a length of 28 s, which means that the deadline is missed when only time redundancy is used and the error is detected by D_8. The schedule depicts the detection operations as thick lines labeled with the operation name. For this example we consider that detections happen in zero time. However, in our implementation, the time needed for the detection operation is calculated as the routing time to bring the droplet to the sensor plus the sensing time. We also consider the waiting time in case the sensor is busy with another detection operation.

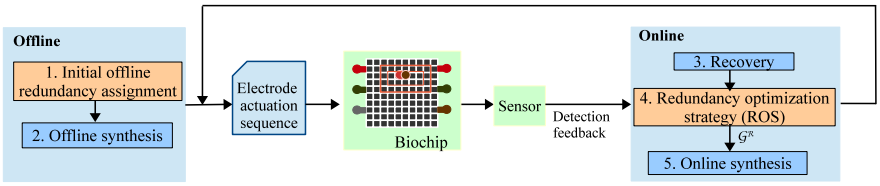
If we use only space redundancy, as in case (b), we miss the deadline in error scenarios (2) and (4). The biochip used in this example has an area of 10×8 electrodes. However, if we use an area of 10×11 electrodes, and also add one extra reservoirs for reagent to the biochip architecture, to parallelize the dispensing, we obtain an application completion time within the required deadline for all error scenarios by using space redundancy only. Our online recovery approach takes into account the available resources when optimizing the redundancy. For a 10×8 architecture, using only space redundancy is not a good option.

In solution (c), using time redundancy for D_6 and space redundancy for D_8 also turned out to be a bad decision, since we miss the deadline in the error scenarios (2) and (4).

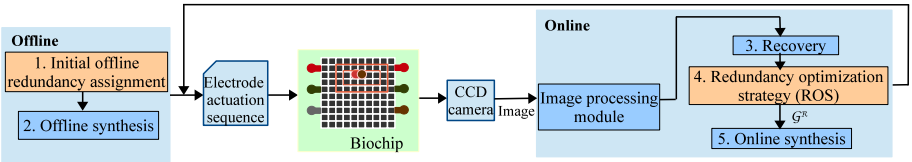
This motivational example shows that (i) using a single fault-tolerance technique is not a good decision and that (ii) we need to find the right combination of time and space redundancy to tolerate the faults in all possible error scenarios, and that (iii) the right decisions depend also on the application and architecture. Our redundancy optimization approach will decide online the introduction of the right combination of fault-tolerance, such that the number of transient errors tolerated is maximized and the application deadline is satisfied. In case our strategy cannot tolerate a specific error scenario, i.e., it cannot complete the application within deadline, we consider that the application has failed.

5 Online error recovery strategy

Figure 8a presents the general strategy of our online recovery approach for the case when a sensor-based detection is used. We discuss the case when a CCD camera-based detection is used in Sect. 7.



(a) Online recovery strategy for sensor-based detection



(b) Online recovery strategy for CCD camera-based detection

Fig. 8 Online error recovery strategy

Our strategy has two components: an offline component consisting of steps 1 and 2, performed at design time, and an online component, steps 3–5, invoked during the execution of the biochemical application. The steps presented in light blue rectangles (see Fig. 8a) are executed on a computer or a microcontroller, whereas the ones in green rectangles are performed on the biochip. Steps 1 and 2 produce offline a fault-tolerant implementation without performing redundancy optimizations that are possible once the error scenarios are known at runtime. Step 1 decides an initial redundancy assignment and the produced fault-tolerant graph is then synthesized during step 2. The initial offline redundancy assignment from step 1 can be decided manually by the designer (as we do in [22]) or by any other method. The same way, step 2 can be implemented using any available synthesis such as the Tabu-Search [37,38] or Simulated Annealing-based [3] implementations. In the experimental results we have used our Redundancy Optimization Strategy (ROS), from Sect. 6 considering a no-faults scenario, to produce the initial redundancy assignment for step 1, and the synthesis from [22] for step 2.

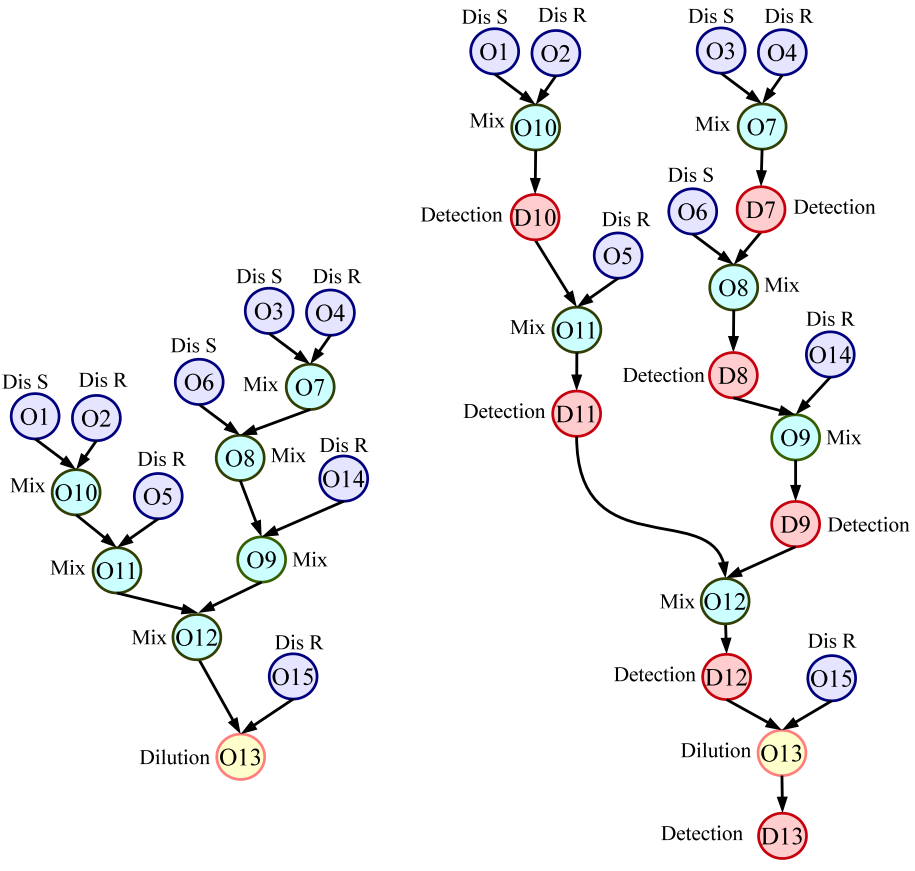
The offline synthesis results are executed on the biochip until a detection operation finishes, when the bioassay execution is stopped and the online component (which is the focus of this paper) is invoked. If an error is detected by the detection operation D_i, we use step 3 to recover. As described in Sect. 3.2, if time redundancy has been previously assigned to D_i, we run the corresponding subgraph R_i to recover from the error detected by D_i. If space redundancy has been assigned to D_i, we use for recovery the redundant droplets produced by R_i^{Space}. Next, at step 4, we run our Redundancy Optimization Strategy (ROS), which optimizes the introduction of detection points and associated redundancy (see Sect. 6).

ROS uses the available information about the current error scenario to optimize the assignment of time and space redundancy for fault-tolerance. Hence, ROS is invoked only when new information about the occurrences of errors is available, that is, after the detection operations (see the arrow labeled "Detection feedback" in Fig. 8a). The fault-tolerant graph $\mathcal{G}^R$, outputted by ROS, is synthesized during step 5, determining a new electrode actuation sequence to be executed on the biochip. The synthesis implementation for step 5 has to be fast, as it is run online and will add an overhead to the execution of the bioassay. Hence, for step 5 we use our List-Scheduling (LS)-based [22] online synthesis as it is able to obtain good quality results in a short time.

5.1 Recovery strategy example

Let us consider the application graph $\mathcal{G}^0$ from Fig. 9a, which is executed on a biochip of 8×7 electrodes, with one dispensing reservoir for the sample $Dis\ S$, and one for the reagent $Dis\ R$, using the module library from Table 1. Detection is performed using a capacitive sensor. During the offline step 1 (see Fig. 8a), the detection operations D_{7-13} are inserted in $\mathcal{G}^0$, after operations O_{7-13}, respectively, as depicted in Fig. 9b. To determine the locations of the detection operations, we use the fault propagation model from [18], as described in Sect. 3.1, considering an error threshold $E_{Thr} = 10$ %. For this example, we assume that, during step 1, time redundancy was assigned for all detection operations. The resulted graph (not depicted for space reasons) with detection operations and corresponding time-redundant subgraphs is given as input to the offline synthesis, which derives the results from Fig. 10a.

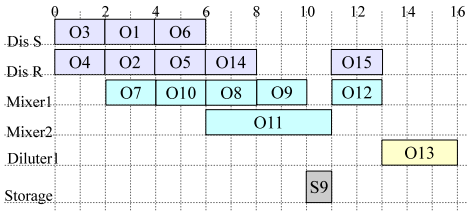
Let us assume that only one transient error occurs during the execution of the application, and it affects operation O_{12}. Hence, at time $t = 4$ s when the detection operation D_7 finishes executing, no error will be detected. We now have the information that D_7 has not detected an error, so we invoke online steps 4 and 5 from our online recovery strategy depicted in Fig. 8a.



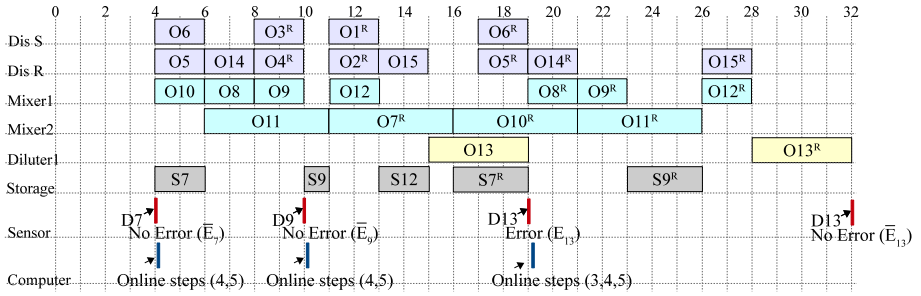
(a) Application graph $\mathcal{G}^0$

(b) $\mathcal{G}^+$ at t = 0 for E_{Thr} = 10%

Fig. 9 Example of recovery strategy—offline step 1



(a) Schedule determined offline



(b) The execution of the application from Fig. 9a when an error occurs in operation O_{12}

Fig. 10 Schedules for execution of application from Fig. 9a

During step 4, we decide when to introduce detection operations and which redundancy techniques to use. In step 5 we synthesize this new implementation online, updating thus the "electrode actuation sequence".

Thus, at time $t = 4$ s when the detection operation D_7 finishes executing, ROS will be called in step 4 and will decide to reduce the number of detection operations to only three (D_9, D_{11} and D_{13} from Fig. 11a), and insert the redundant subgraphs R_9^{Space} for D_9, R_{11}^{Space} for D_{11} and R_{13}^{Time} for D_{13}. The details of how ROS works are presented in the next section. In this example we show that ROS has decided increase the value of the error threshold to $E_{Thr} = 12\%$ and thus remove the detection operation D_{12} (see Sect. 6.1 for a discussion on the advantages of such a decision). ROS will output the graph $\mathcal{G}^R$ with the new detection operations and redundant subgraphs from Fig. 12.

The synthesis in step 5 takes as input $\mathcal{G}^R$ and derives a new implementation. For step 5, we have used our List Scheduling (LS)-based synthesis from [22] to perform binding, placement, routing and scheduling. Part of the resulted schedule is presented in Fig. 10b, between $t = 4$ and $t = 10$ s. In Fig. 10b, which depicts the execution of the application at runtime, the overhead due to the execution of the online steps is represented as a blue line under the row labeled "Computer". The redundant operations, part of the inserted redundant subgraphs, are marked O_i^R in the schedule (e.g., O_1^R).

The new implementation continues to execute until the next detection operation finishes. As depicted in Fig. 10b, the online steps 4 and 5 are invoked again at $t = 10$ s, after detection D_9 finishes executing. Part of the new resulted schedule is depicted in Fig. 10b, between $t = 10$ and $t = 19$ s. As mentioned, we have assumed that a transient error will affect O_{12}. (Note that the errors are unpredictable.) The error in O_{12} is detected by D_{13} (since the error will propagate) at $t = 19$. This will trigger the online recovery step 3 of our strategy, followed by steps 4 and 5. The application completes in 32 s and has tolerated the transient error in O_{12}.

6 Redundancy optimization strategy (ROS)

Our Redundancy Optimization Strategy (ROS) is presented in Fig. 13. It takes as input the detection operation D which triggered it, the graph $\mathcal{G}'$, the biochip architecture $\mathcal{A}$, the estimated number of faults k^0, the number k of faults occurred so far and the current time t. $\mathcal{G}'$ is the currently executing application graph, from which we have removed the operations which have finished executing, the previously decided detection operations and their associated recovery subgraphs.

ROS has three components. First, it decides where to insert detection operations, lines 1–3 in Fig. 13 and discussed in Sect. 6.1. Second, for each inserted detection operation, ROS decides between time and space redundancy, see Sect. 6.2. ROS prefers space redundancy for important operations as long as there is enough area for the corresponding redundant subgraph (lines 4–15), and uses time redundancy for the rest (lines 16–19). Third, ROS has to determine, for each redundancy scheme introduced, the redundant subgraph R_i, see the discussion in Sect. 3.2. This is done in lines 8 and 17 in Fig. 13, as discussed in Sect. 6.3.

Based on the error information after the detection operation and on the current configuration (redundant droplets available), the goal is to minimize the resources used by redundancy (slack time and area) such that the number of tolerated transient errors is maximized. ROS produces a new application graph $\mathcal{G}^R$, with updated detection points and fault-tolerance, which is passed to the online synthesis in step 5, Fig. 8a.

6.1 Deciding the detection operations

The detection operations and the associated redundancy are required for fault-tolerance. However, redundancy introduces delays in the application execution in case there are no faults. In case of faults, it is important to detect and recover from them as soon as possible, so that no time is wasted. Researchers have used the fault analysis from Sect. 3.1, based on a designer-specified error threshold E_{Thr}. To decide where to introduce the detection operations, a given E_{Thr} assumes a number of faults k^0 that may happen during a given time (this is similar to the fault rate of VLSI circuits). In order to decide where to insert the detection operations, ROS first adjusts the value of E_{Thr} according to the actual number of faults. Then, ROS uses the fault analysis from Sect. 3.1 with the new E_{Thr} to decide the detections. This is especially important for biochips used in applications that require monitoring over a long time, such as bioterrorism, environment and water monitoring.

The threshold E_{Thr} is adjusted in the **AdjustErrorThreshold** function in line 1, Fig. 13. The function receives the number of faults k^0 expected over a given time period, the number of faults k that have been detected so far, and the time t. The time period is specified as a multiple of the application deadline (which is also its period, for monitoring applications) and the time t is relative to the current invocation of the application. The number k^0 represents the number of errors expected until t and it is given by the accuracy requirements of the application. We assume that the faults are uniformly distributed in time. This assumption is used only to adjust E_{Thr} and does not affect our ability to provide fault-tolerance. In case k is less than expected, E_{Thr} is increased proportionally, allowing less detection operations to be inserted. Otherwise, E_{Thr} is decreased, resulting in more detection operations. Note that our online strategy performs recovery from any transient error detected within the application deadline. In that context, we can tolerate more errors than the expected number of errors k^0.

Considering the example from Fig. 9a, and that no fault happened so far, at time $t = 4$ s we adjust E_{Thr} from 10 to 12 %.

We then call the function **DetermineDetectionOperations** using the new E_{Thr} values, line 2. The function uses the fault analysis from Sect. 3.1 to calculate the fault limits for each operation in $\mathcal{G}'$. For the example in Fig. 11a, we conclude that operations O_9, O_{11} and O_{13} exceed the threshold error $E_{Thr} = 12\ \%$. It follows that for operations O_9, O_{11} and O_{13} we will need the detection operations D_9, D_{11} and D_{13} which are returned as a queue Q. Finally, the function **InsertDetections** from line 3 inserts the detection operations from Q into the graph $\mathcal{G}'$ (see the graph from Fig. 11a).

6.2 Redundancy optimization strategy

For each operation D_i in Q, ROS has to decide the associated redundant subgraph R_i, and insert it into the current graph $\mathcal{G}^R$. Section 3.2 has discussed the trade-offs between time and space redundancy. Our heuristic strategy in ROS is to introduce space redundancy (because it saves time at the expense of area) only if the extra area used does not lead to greater delays (because regular operations do not have space to execute on the biochip). For the cases when ROS decides that space redundancy is not appropriate, it introduces time redundancy instead.

Thus, in the repeat loop (lines 6–15 from Fig. 13), we decide where to introduce space redundancy. We consider every operation D_i in the queue Q. At line 4, we prioritize the order in which we visit the detections according to a priority function $Priority(D_i)$. We want to prioritize those detection operations (1) for which we predict that an error is more likely to occur and (2) whose redundant droplets produced by space redundancy can be reused by operations on the critical path[1] of the application graph, in case the predicted error does not occur. These two cases are captured by the two terms of the following equation, where a and b are weights given by the designer:

$$Priority(D_i) = a \times E_i + b \times RF_i, \tag{1}$$

(1) Regarding the first term, we assume that an error is more likely to occur if the fault limit E_i (first term) of the operation D_i is higher.

(2) The second term is calculated in the following way: in case an error does not occur we would like to reuse the correctly sized redundant droplets produced by the subgraph R_i^{Space}. The completion time $\delta_{\mathcal{G}}$ of the current graph $\mathcal{G}^R$ is determined by the critical path of $\mathcal{G}^R$. To reduce $\delta_{\mathcal{G}}$, we prefer that the droplets from R_i^{Space} are reused by operations on the critical path. This is captured by the reusability factor RF_i in the second term. The reusability factor RF_i is given by the cumulative execution time T_i over all the operations that can use the droplet produced by R_i^{Space}. For a fair comparison to E_i, which is a percentage, we obtain RF_i by dividing T_i to the execution time of the critical path. For example, let us consider the detection operation D_9. The droplet produced by R_9^{Space} can be used by operation O_9, in case an error is detected by D_9. Otherwise (i.e., in case D_9 does not detect an error, the droplet produced by R_9^{Space} can be stored and used for recovery from an error in operation O_{12} or in operation O_{13} (Fig. 11a). Note that if an error occurs in O_{13}, operation O_{12} has to be re-executed, and thus it can use the redundant droplet produced by R_9^{Space}. The total execution time T_9, calculated for operations O_9, O_{12} and O_{13} is of 8 s. The critical path execution time is 12 s, so we obtain the reusability factor $RF_9 = 0.66$, as shown in Fig. 11b.

Considering $a = 0.4$ and $b = 0.6$, we obtained for detections D_9, D_{11} and D_{13} from Fig. 11a, the values from Fig. 11b. Detection D_{13} has the lowest priority.

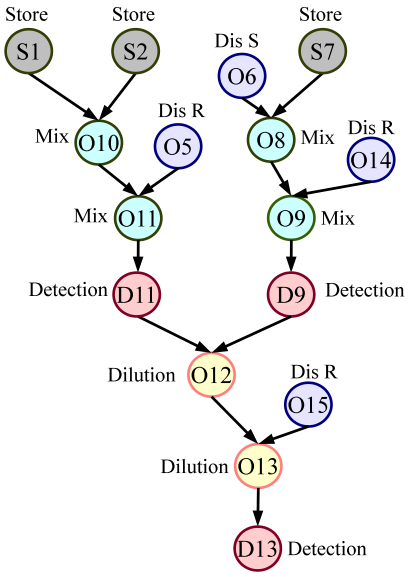
[1] The critical path is defined as the longest path in the graph [39], between the root and the leaf nodes.

Next, we decide for each detection operation, prioritized as explained previously, whether we introduce space or time redundancy. We use a List Scheduling (LS)-based online synthesis, as discussed in Sect. 4.1. LS uses a priority function to select among operations which are ready for execution (this is different from the priority function we use for the detection operations, $Priority(D_i)$). As discussed, space redundancy may introduce delays in the application completion because the execution of the redundant operations can take away area from the other operations. To avoid this situation, our approach with ROS is to use a lower LS-priority function for the operations O_i^R from the space redundant subgraph R_i^{Space} compared to regular operations. However, even if executed with lower priority, the redundant operations O_i^R produce intermediate droplets that need to be stored on the biochip for later use. Storing the intermediate droplets can take away area from the other operations, causing delays in the application completion time. We can determine exactly these delays by running an online synthesis, such as the one in [22], which determines if the introduction of space redundancy delays the application completion time. However, the synthesis takes time and has to be run for each detection operation.

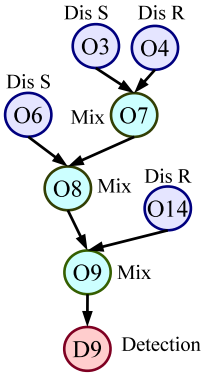
Instead, our heuristic with ROS is to quickly estimate these delays (i.e., caused by storage of redundant droplets) without performing a synthesis, as follows. The repeat loop (lines 6–15 in Fig. 13) removes each detection operation D_i from the head of the priority-sorted queue Q. For each such D_i, our approach calculates the required area r_{area} to store the redundant droplets produced by R_i^{Space} (line 9). The required area r_{area} is calculated by traversing R_i^{Space} and determining the maximum number of operations that can execute simultaneously (also known as the maximum width of a tree). Note that the execution of R_i^{Space} can be interrupted at any time, since the redundant operations in R_i^{Space} have lower priority. For that reason we consider that the maximum number of droplets that need to be stored at a time is the maximum number of intermediate droplets produced by R_i^{Space} at the same time. For example, the r_{area} for R_9^{Space} in Fig. 11c is of $2 \times 9 = 18$ electrodes, since maximum two operations can run in parallel and nine electrodes are needed to store each droplet (see the "Store" operation in Table 1).

Next, our heuristic determines the available area a_{area} on the biochip and if a_{area} can accommodate r_{area}, then it introduces space redundancy for detection D_i. We estimate the maximum time interval $[t_i^{start}, t_i^{stop}]$ during which R_i^{Space} will be executed. The start time t_i^{start} is given by the earliest time t when R_i^{Space} can start executing. For example, R_9^{Space} in Fig. 11c cannot start executing before $t_9^{start} = 6$ s, when the reservoir $Dis\ S$ is free to be used. The stop time t_i^{stop} is calculated starting from the time moment when the detection D_i is executed and adding the critical path execution time for R_i^{Space}. For R_9^{Space} (Fig. 11c) $t_9^{stop} = 18$ s, obtained by adding the critical path execution time of R_9^{Space}, which is 8 s, to the time moment $t = 10$ s, when detection D_9 finished. The critical paths are determined offline for every relevant operation and are adjusted online. We use the **AreaFunction** to calculate the available area a_{area} for the determined time interval $[t_i^{start}, t_i^{stop}]$ (line 10 in Fig. 13). If there is enough available area, condition checked in line 11, ROS decides to introduce space redundancy for D_i.

The repeat loop (lines 6–15 in Fig. 13) terminates when there is not enough available area, or if the priority-sorted queue Q is empty. Next, if Q is not empty, ROS assigns time redundancy for all the remaining detection operations (lines 16–19). In our example, there is enough storage area for R_9^{Space} and R_{11}^{Space}, so space redundancy is assigned for D_9 and D_{11}. The remaining available area is not large enough to accommodate R_{13}^{Space}, therefore time



(a) Graph at t = 4 for $E_{Thr} = 12$



(c) Recovery subgraph R_9^{Space} for D_9

D_i	E_i	RF_i	Priority	Redundancy
D_9	0.12	0.66	0.44	space
D_{11}	0.122	0.66	0.44	space
D_{13}	0.172	0.33	0.26	time

(b) Priority calculation for detection operations

Fig. 11 Online redundancy assignment at t = 4 for the application graph from Fig. 9a

redundancy is assigned to D_{13}. The space and time-redundant subgraphs are inserted in the graph (lines 12 and 18) obtaining, for the graph in Fig. 9a, the graph $\mathcal{G}^R$ depicted in Fig. 12.

6.3 Generating the recovery subgraph

The algorithm in Fig. 4 determines online the recovery subgraph R_i for a detection operation D_i. The recovery subgraph R_i contains the redundant operations needed to produce the correct droplets for the operation O_i. The subgraph R_i is inserted in the graph by ROS, either using space redundancy (line 12 in Fig. 13) or time redundancy (line 18 in Fig. 13). For example, the recovery subgraph R_9^{Space}, for detection operation D_9, is illustrated in Fig. 11c.

Starting from the considered detection D_i, the algorithm uses the breadth-first search (BFS) technique to traverse the graph (line 3 in Fig. 4). All explored operations are inserted in the recovery subgraph R_i. The search stops when no more operations can be inserted, i.e., the root nodes (which are dispensing operations in our case) are reached. The subgraph R_i is updated online by taking into account the redundant droplets stored on the biochip (lines 9–22). These droplets can be by-product droplets intended for discarding (e.g., produced by a dilution operation) or droplets generated by the redundant operations inserted for recovery. The list L_{stg} keeps track of the by-product droplets and of the ones that come from previous redundant operations. These steps are done offline and the resulted subgraphs are stored for each operation, to be used by ROS online. The subgraph R_i is traversed using **BFS**, see the repeat loop (lines 9–22). For each explored operation O_i, the algorithm checks the list of redundant droplets L_{stg}. In case a matching droplet n is found for O_i, the subgraph R_i is pruned (line 14) and L_{stg} is updated (line 15). If no matching droplet is found in the storage units for operation O_i, then all the unexplored predecessors of O_i are enqueued to be explored. The algorithm stops when there is no operation to be explored.

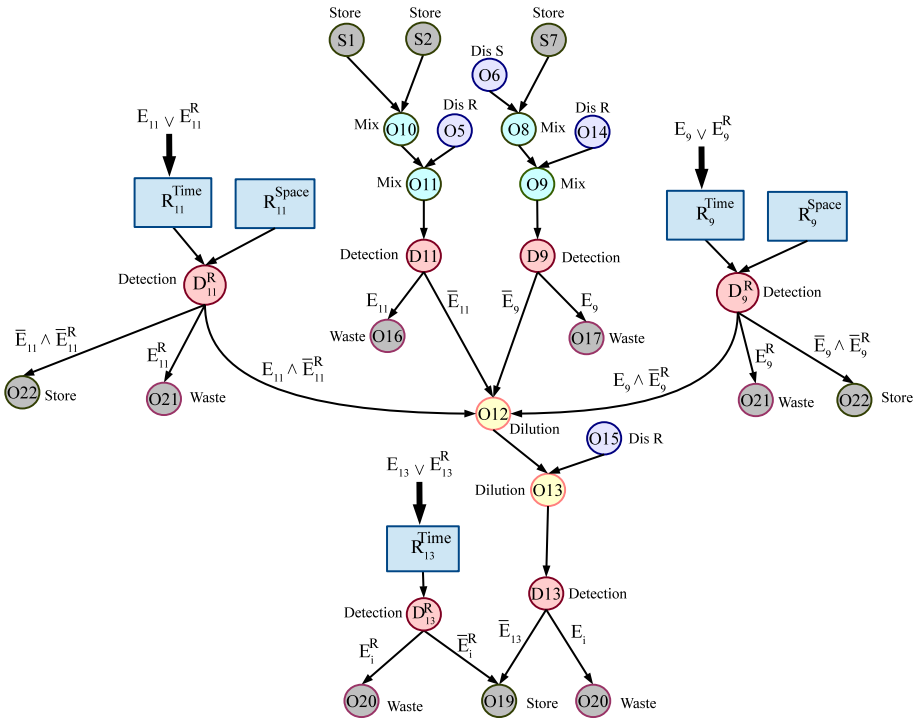


Fig. 12 Graph $\mathcal{G}^R$ produced by ROS at t = 4 for the application graph from Fig. 9a

In the example from Fig. 3a, L_{stg} consists of the unused droplets produced by dilution operations O_3 and O_8. In this case, the algorithm uses the stored droplets and prunes the recovery subgraph R_{11}. Consequently, the size of R_{11} is reduced from 11 operations (Fig. 3b) to 7 operations (Fig. 3c), leading to a shorter recovery time. The structure of the recovery subgraph depends on the current error scenario, as redundant droplets can result from previous recovery operations.

7 Error recovery strategy with CCD camera-based detection system

The CCD camera-based detection system is proposed in [25] as an error detection alternative to capacitive sensors. Using a CCD camera, images of the droplets on the biochip are captured periodically and analyzed, using pattern matching, in order to locate the position and the size of the droplets. The main advantage of using a CCD camera-based detection system over a sensor-based detection, is that, since the detection is performed simultaneously and continuously, the error is detected immediately when it occurred. When using a sensor, the detection operations are scheduled at specific times and, therefore, the error can be detected long after its occurrence. The online recovery steps are taken as soon as the error is detected. Hence, when using a sensor, the recovery is delayed, resulting in longer completion times. Moreover, the use of a CCD-camera based detection system eliminates the need for routing the droplets to a specific location, or to wait in case there are not enough available sensors. ROS is able to optimize the introduction of redundancy because it makes use of the information

Fig. 13 Redundancy optimization strategy (ROS)

ROS$(D, \mathcal{G}', \mathcal{A}, k^0, k, t)$
1: E_{Thr} = AdjustErrorThreshold(k^0, k, t)
2: Q = DetermineDetectionOperations($\mathcal{G}', E_{Thr}$)
3: $\mathcal{G}^R$ = InsertDetections($\mathcal{G}'$,Q)
4: Prioritize(Q)
5: $AreaFunction$ = GetAreaFunction($\mathcal{G}', \mathcal{A}, t$)
6: **repeat**
7: D_i = Head(Q)
8: R_i^{Space} = SpaceRedundantSubgraph(D_i, $\mathcal{G}'$)
9: r_{area} = RequiredArea(R_i^{Space})
10: a_{area} = AvailableArea($AreaFunction, R_i^{Space}$)
11: **if** $a_{area} \geq r_{area}$ **then**
12: Insert(R_i^{Space}, $\mathcal{G}^R$)
13: UpdateArea($AreaFunction$, R_i^{Space})
14: **end if**
15: **until** $Q = \emptyset$ or $a_{area} < r_{area}$
16: **for each** remaining D_i in Q **do**
17: R_i^{Time} = TimeRedundantSubgraph(D_i, $\mathcal{G}^R$)
18: Insert(R_i^{Time}, $\mathcal{G}^R$)
19: **end for**
20: **return** $\mathcal{G}^R$

about fault occurrences. Both situations are important for ROS: if an error has happened and if an error has not occurred. With the setup in Fig. 8b, ROS would be called only if faults are occurring. Our strategy is to introduce in the application graph places where ROS would be called, so it could take informed decisions about how to allocate redundancy. We use the same approach we have used to insert the detection operations in lines 3 and 18 in Fig. 13, see Sect. 6.2, but instead of detection operations we introduce "triggering" operations, which will invoke ROS at runtime.

The general strategy of our online recovery approach when using a CCD camera-based detection system is presented in Fig. 8b. Images are captured continuously throughout the execution of the bioassay. When the image processing module signals an error, the execution of the bioassay is interrupted and the online steps 3, 4 and 5 are executed. Note that using a CCD camera-based detection system does not require introduction of detection operations in the application graph, as is the case with a capacitive sensor. Hence, ROS can be triggered during any operation in step 4, as soon as an error is detected after the recovery in step 3.

Considering the example discussed in Sect. 5.1, if a CCD camera-based detection system is used, the application completes in 29 s, which is 9.3 % faster compared to using capacitive sensors (Fig. 10b). The execution of the application at runtime when using a CCD camera-based detection is depicted in Fig. 14. The reduction in completion time comes from detecting the error when it occurred, during O_{12}, at $t = 14$ s. When a capacitive sensor is used, the detection is scheduled at $t = 19$ s (see Fig. 10b), so the error is detected with a delay. For the situations when the additional equipment for image capturing and processing is available, and portability is not required, a detection system based on CCD cameras, provides the fastest results at the moment. However, our proposed ROS does not depend on a specific detection method and can be integrated with any available technology.

8 Experimental results

For experiments we used three real-life applications: (1) the sample preparation for plasmid DNA (PDNA, 19 operations); (2) the colorimetric protein assay (CPA, 103 operations) and

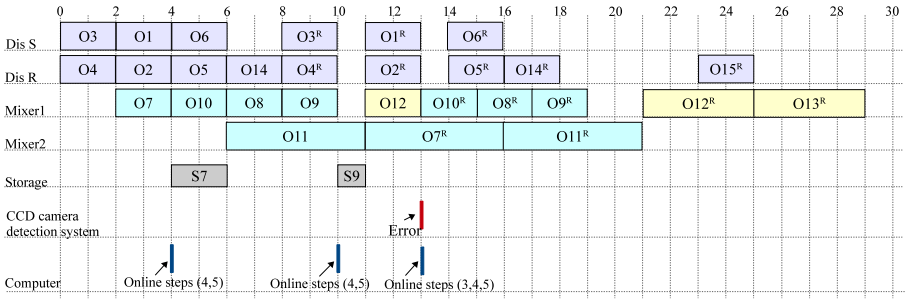


Fig. 14 The execution of the application from Fig. 9a, with CCD camera-based detection system

(3) the interpolation dilution of a protein (IDP, 71 operations). The application graphs and the descriptions of the bioassays can be found in [21] for PDNA and in [18] for CPA and IDP. The deadlines for PDNA, CPA and IDP are $d_{PDNA} = 60$ s, $d_{CPA} = 300$ s, $d_{IDP} = 200$ s, respectively. The algorithms were implemented in Java (JDK 1.6) and run on a computer with Intel Celeron 560 CPU at 2.13 GHz and 2 GB of RAM. Both the simulation of the application execution and the online error recovery strategy were executed on the mentioned hardware. We used the experimentally determined module library from Table 1 [32]. We have performed three sets of experiments. In the first two sets we have considered only a capacitive sensor for detection, i.e., not a CCD camera system. In all experiments we have approximated the routing overhead as the Manhattan distance between the top-left corners of the modules.

In the first set of experiments we were interested to determine if is is important to use a combination of redundancy techniques (i.e., time and space redundancy) and if ROS is able to optimize their allocation. Hence, we have compared (a) our redundancy optimization approach ROS with two cases where we have used (b) only time redundancy for recovery, called TIME, and (c) only space redundancy (SPACE). The recovery subgraphs for case (b) and (c) were assigned statically offline for all the detection operations. For this set of experiments, we used 3 different biochips (column 1 in Table 3), with sizes of 7×7, 8×9 and 10×10 electrodes. Next to the sizes, we also present in parentheses the numbers of reservoirs for the three reagents (respectively $R1$, $R2$ and $R3$) used by PDNA, see [21] for details. The techniques are compared in terms of the application completion time $\delta_{\mathcal{G}}$ obtained for PDNA. Since a particular error can be favorable to a certain redundancy technique, in the interest of a fair comparison, we have generated randomly 50 error scenarios, and we used for comparison the average value of $\delta_{\mathcal{G}}$ obtained over all scenarios.

Related work has considered an occurrence of maximum two faults during the experiments [19]. Hence, in our experiments we also consider a maximum number of faults $k = 2$ in order have a fair comparison to earlier work. However, in reality more than 2 faults may occur during the execution of the biochemical application [40]. Such situations occur mostly because of the randomness in the fabrication process, when the thickness of the insulator and the hydrophobic layers may vary from one electrode to another. Consequently, the electrowetting forces generated on the surface of two neighboring electrodes may not be exactly the same—the main cause of an erroneous split operation.

Thus, we have simulated the execution of PDNA on each of the three biochips, and we have randomly inserted $k = 1$ and 2 errors in the operations. We show the obtained average (avg.) $\delta_{\mathcal{G}}$ and the mean deviation (dev.) in Table 3, columns 2, 3 and 5 for the three cases (a)–(c). The reported $\delta_{\mathcal{G}}$ times take into account the runtime overhead required by re-synthesis

Table 3 Comparison between recovery techniques for PDNA

Arch.	(a) $\delta_{\mathcal{G}}^{ROS}$ (s)	(b) $\delta_{\mathcal{G}}^{TIME}$ (s)	Improvement (%)	(c) $\delta_{\mathcal{G}}^{SPACE}$ (s)	Improvement (%)
7×7	Avg. 34.08	Avg. 56.28	39.4	Avg. 64.07	46.8
(2, 1, 2)	Dev. 2.84	Dev. 8.46		Dev. 14.51	
8×9	Avg. 32.68	Avg. 55.61	41.2	Avg. 58.72	44.3
(2, 1, 2)	Dev. 0.98	Dev. 8.97		Dev. 10.23	
10×10	Avg. 27.5	Avg. 54.8	49.8	Avg. 57.92	52.5
(2, 1, 2)	Dev. 3.13	Dev. 7.84		Dev. 9.55	

(for all cases) and the runtime of the redundancy optimization, performed only in the case of ROS. The mean deviation (dev.) is calculated as the average over the absolute values of deviations from the average completion time (avg.). In columns 4 and 6, we show the percentage improvement of ROS over TIME and SPACE, calculated as: $\frac{\delta_{\mathcal{G}}^{TIME} - \delta_{\mathcal{G}}^{ROS}}{\delta_{\mathcal{G}}^{TIME}} \times 100$ and $\frac{\delta_{\mathcal{G}}^{SPACE} - \delta_{\mathcal{G}}^{ROS}}{\delta_{\mathcal{G}}^{SPACE}} \times 100$.

From Table 3 we see that ROS, which uses an optimized combination of space and time redundancy, is able to obtain much better results than using a single form of redundancy, TIME or SPACE. Compared to TIME, ROS leads to an improvement of 39 % for the 7×7 biochip, 41 % for 8×9 and 49 % for 10×10 (see column 4). The improvement over SPACE is 46, 44 and 52 %, respectively. Better results were obtained for larger biochip areas, as ROS uses the available space to optimize the introduction of space redundancy and reduce the recovery time. All the considered areas are, however, too small to use space redundancy exclusively. As the biochip area increases, from 7×7 to 10×10, all techniques benefit of the extra area and use it to improve $\delta_{\mathcal{G}}$, hence the decrease in $\delta_{\mathcal{G}}$ as area increases. However, the percentage improvement between ROS and the others gets larger, as ROS is better at exploiting the extra area. The 10×10 area is still too small to use space redundancy exclusively, hence SPACE gives worse results than ROS. Regarding the deadline, all solutions obtained with ROS meet the deadline, i.e., $\delta_{\mathcal{G}}^{ROS} \leq d_{PDNA}$, whereas the deadline is satisfied only in 56 % of cases for TIME and 49.4 % of cases for SPACE. This experiment shows that by using our proposed ROS, which decides online between the introduction of time and space redundancy, we obtain better results compared to using a single redundancy technique.

In the second set of experiments we were interested to compare ROS to the related work. Thus, we compared the completion time $\delta_{\mathcal{G}}^{ROS}$ obtained by ROS with the $\delta_{\mathcal{G}}^{DICT}$ obtained by using the previously proposed dictionary-based error recovery (DICT) [19]. DICT determines offline the recovery needed for an error and the corresponding changes to the electrode actuation sequence for the operations, then, it stores the results in a dictionary, to be used online, when an error is detected. Hence, DICT has negligible runtime overhead for applying the recovery. In contrast, ROS determines both the required recovery and the changes to the electrode actuation sequence (what we call re-synthesis) online, during the execution of the biochemical application. We ran experiments for CPA and IDP, using the same error scenarios and biochip configuration as in [19]. The results are presented in Table 4 for CPA, and in Table 5 for IDP. The completion time $\delta_{\mathcal{G}}^{DICT}$ is presented in column 2, and $\delta_{\mathcal{G}}^{ROS}$ in column 3.

Table 4 Comparison of dictionary-based error recovery [19] and ROS for CPA

Errors (ops.)	$\delta_{\mathcal{G}}^{DICT}$ (s)	$\delta_{\mathcal{G}}^{ROS}$ (s)	CPU time (s)	Improvement (%)
Dlt_{39}	228	212.21	0.98	6.92
Dlt_{12}, Dlt_{31}	220	192.19	0.9	12.64
DsB_4, Dlt_{14}	219	192.25	1.12	12.21
Dlt_{21}, Mix_5	223	219.26	1.06	1.67

Table 5 Comparison of dictionary-based error recovery [19] and ROS for IDP

Errors (ops.)	$\delta_{\mathcal{G}}^{DICT}$ (s)	$\delta_{\mathcal{G}}^{ROS}$ (s)	CPU time (s)	Improvement (%)
Dlt_8, Dlt_{16}	208	161	1.7	22.5
Dlt_2, Dlt_{29}	212	175.86	1.5	17
Dlt_{19}, DsB_{23}	207	163.77	0.5	20.8
Dlt_{16}, Dlt_{18}	209	163.65	0.4	21.7

$\delta_{\mathcal{G}}^{ROS}$ contains the runtime execution overhead of the online steps of our strategy (see Fig. 8a). This overhead is also reported separately in the tables in column 4. These runtimes are cumulative, a summation for all invocations of ROS in the given scenario, and are measured on a typical PC, which is used to control the biochip. As the results in Tables 4 and 5 show, our approach (ROS) is able to obtain much better results compared to the related work DICT (more than 20% reduction for a third of the cases). The percentage improvement of ROS over DICT is shown in the last column in the two tables. The improvement of our proposed online redundancy approach comes from the optimized use of recovery techniques employed. For example, for IDP, where a larger biochip area is available for operations, ROS has used space redundancy for carefully selected operations, which trades off area for time, in order to improve the results.

As mentioned in the problem formulation, with ROS we are interested to maximize the number of transient errors tolerated within the application deadline. An application tolerates the faults if the deadline is satisfied, i.e., $\delta_G \leq d_G$, in all the fault scenarios. Thus, in the last set of experiments we were interested to find out if ROS can synthesize online a fault-tolerant implementation which meets the deadline as the number of faults k increases. We ran the experiments for all three benchmarks: PDNA, IDP and CPA and we present the results in Table 6. The biochip sizes used for each application is presented in column two. Next to the sizes, we also present in parentheses the numbers of reservoirs for the sample, buffer and reagents. We have generated a large number of error scenarios covering possible combinations of k faults and operations. We ran the experiments using both detection methods presented in the paper: the sensor-based detection and the CCD camera-based detection system. The $\delta_{\mathcal{G}}^{ROS}$ values reported are the shortest completion time (min), the longest completion time (max), the average completion time (avg.) and the mean deviation (dev.) over all the simulation runs. We have generated the error scenarios considering the size of the applications and the number of errors: between 50 and 100 error scenarios for $k = 1$, between 500 and 1000 error scenarios for $k = 2$ and between 1500 and 2000 error scenarios for $k = 3$. The results for $k = 1, 2$ and 3 are presented in columns three, four and five, respectively, for the case when a capacitive sensor-based detection is used, and in columns six, seven and eight, for the case when CCD camera-based detection is used. As we see from the table, ROS is

Table 6 ROS results for k = 1, 2 and 3 faults

App. ops.	Arch.	Capacitive sensor			CCD		
		$\delta_{\mathcal{G}}^{ROS}$ (s) k = 1	$\delta_{\mathcal{G}}^{ROS}$ (s) k = 2	$\delta_{\mathcal{G}}^{ROS}(s)$ k = 3	$\delta_{\mathcal{G}}^{ROS}(s)$ k = 1	$\delta_{\mathcal{G}}^{ROS}$ (s) k = 2	$\delta_{\mathcal{G}}^{ROS}$ (s) k = 3
PDNA (19)	7 × 7 (1, 2, 2)	Min 30.24	Min 30.28	Min 32.25	Min 25.14	Min 25.14	Min 25.22
		Max 37.25	Max 37.25	Max 37.4	Max 33.64	Max 34.42	Max 37.19
		Avg. 32.62	Avg. 33.33	Avg. 34.62	Avg. 29.46	Avg. 30.38	Avg. 31.13
		Dev. 2.42	Dev. 2.66	Dev. 2.48	Dev. 1.52	Dev. 1.71	Dev. 1.98
IDP (71)	9 × 9 (1, 2, 2)	Min 159.66	Min 159.75	Min 160.71	Min 139.61	Min 139.7	Min 141.9
		Max 166.63	Max 177.61	Max 182.66	Max 169.11	Max 174.98	Max 178.98
		Avg. 161.97	Avg. 166.52	Avg. 168.04	Avg. 157.27	Avg. 159.87	Avg. 160.27
		Dev. 2.57	Dev. 2.87	Dev. 3.61	Dev. 6.12	Dev. 7.02	Dev. 6.28
CPA (103)	11 × 11 (1, 2, 2)	Min 192.65	Min 192.8	Min 213.38	Min 192.53	Min 192.6	Min 194.06
		Max 219.78	Max 219.93	Max 244.95	Max 215.74	Max 218.61	Max 236.79
		Avg. 198.69	Avg. 209.71	Avg. 219.61	Avg. 197.72	Avg. 207.03	Avg. 217.68
		Dev. 8.65	Dev. 9.03	Dev. 3	Dev. 5.12	Dev. 8.32	Dev. 6.88

able to successfully tolerate an increasing number of faults, producing online fault-tolerant implementations which meet the deadline in all cases (the maximum value of $\delta_{\mathcal{G}}^{ROS}$ is less than the deadlines of the respective benchmarks). The redundancy required for fault-tolerance and the runtime execution of ROS will introduce an overhead. However, it is important to notice that $\delta_{\mathcal{G}}^{ROS}$ increases slowly with k, which means that ROS can successfully tolerate an increasing number of faults. This is because ROS is able to use the fault occurrence information at runtime to optimize the introduction of redundancy, such that the delays on the application completion time $\delta_{\mathcal{G}}$ are minimized. It follows that it is important to use an online redundancy optimization and re-synthesis approach if we want to have fault-tolerant biochip implementations.

Finally, in Table 6 we see the difference between the two sensor setups: using a capacitive sensor, which requires the introduction of detection operations (the columns labeled "sensor"), versus using an imaging CCD camera-sensor which can instantly detect an error, the columns labeled "CCD". As expected, using a CCD camera-sensor leads to better results, because the errors are detected immediately. Our ROS approach can use both setups, and is able to intelligently introduce the detection operations required by the capacitive sensor setup, reducing its inherent delays.

9 Conclusions

In this paper we have presented an online redundancy optimization strategy (ROS) for the synthesis of fault-tolerant biochemical applications. We have addressed digital microfluidic biochips, where the liquids are manipulated using droplets. We have taken into account the parametric faults which can result in operation variability, such as volume variations. The main features of ROS are that it uses a combination of time redundancy (re-executing operations) and space redundancy (producing redundant correct droplets before we know

an error will occur), and is able to optimize at runtime their use based on the actual fault occurrences.

We have also proposed a biochemical application model which captures the sensing operations needed to detect an error, and the operations that have to be executed for recovery. The error detection operations are introduced at runtime based on an fault propagation analysis, our model and the amount of error occurrences. The experiments performed on three real-life case studies show that our redundancy optimization strategy can be successfully used to tolerate transient faults in time-sensitive biochemical applications. In the experiments, we show that our strategy provides faster recovery than the related work. Because our approach can take advantage of the available resources, such as biochip area and devices, we obtained reduced application completion times. Hence, our strategy can be used even in the case of reduced-size systems, such as biochips with reduced area and fewer reservoirs.

References

1. Mukhopadhyay R (2009) Microfluidics: on the slope of enlightenment. Anal Chem 81(11):4169–4173
2. Mark D, Haeberle S, Roth G, von Stetten F, Zengerle R (2010) Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. Chem Soc Rev 39(3):1153–1182
3. Chakrabarty K, Su F (2006) Digital microfluidic biochips: synthesis, testing, and reconfiguration techniques. CRC, Boca Raton
4. Srinivasan V, Pamula VK, Fair RB (2004) An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids. Lab Chip 4(4):310–315
5. Chakrabarty K, Fair RB, Zeng J (2010) Design tools for digital microfluidic biochips: toward functional diversification and more than Moore. IEEE Trans Comput-Aided Des Integr Circuits Syst 29(7):1001–1017
6. Pollack MG, Fair RB, Shenderov AD (2000) Electrowetting-based actuation of liquid droplets for microfluidic applications. Appl Phys Lett 77(11):1725–1726
7. Fair R (2007) Digital microfluidics: is a true lab-on-a-chip possible? Microfluid Nanofluidics 3(3):245–281
8. Huang T-W, Yeh S-Y, Ho T-Y (2010) A network-flow based pin-count aware routing algorithm for broadcast electrode-addressing ewod chips. In: proceedings of the international conference on computer-aided design, pp. 425–431
9. Maftei E, Pop P, Madsen J (2012) Routing-based synthesis of digital microfluidic biochips. Des Autom Embed Syst 16(1):19–44
10. Huang J-D, Liu C-H, Lin H-S (2013) Reactant and waste minimization in multitarget sample preparation on digital microfluidic biochips. IEEE Trans Comput-Aided Des Integr Circuits Syst 32(10):1484–1494
11. Grissom D, Brisk P (2012) Fast online synthesis of generally programmable digital microfluidic biochips. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. pp 413–422
12. Roy P, Rahaman H, Giri C, Dasgupta P (2012) Modelling, detection and diagnosis of multiple faults in cross referencing dmfbs. In: international conference on informatics, electronics & vision, pp 1107–1112
13. Xu T, Chakrabarty K (2009) Fault modeling and functional test methods for digital microfluidic biochips. IEEE Trans Biomed Circuits Syst 3(4):241–253
14. Alistar M, Pop P, Madsen J (2013) Application-specific fault-tolerant architecture synthesis for digital microfluidic biochips. In: Proceedings of the 18th Asia and South Pacific design automation conference, pp 794–800
15. Rose D (1999) Microdispensing technologies in drug discovery. Drug Discov Today 4(9):411–419
16. Kotchoni SO, Gachomo EW, Betiku E, Shonukan OO (2003) A home made kit for plasmid DNA mini-preparation. Afr J Biotechnol 2(4):88–90
17. Alistar M, Maftei E, Pop P, Madsen J (2010) Synthesis of biochemical applications on digital microfluidic biochips with operation variability. In: IEEE symposium on design. Test, integration and packaging of MEMS/MOEMS, pp 350–357
18. Zhao Y, Xu T, Chakrabarty K (2010) Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip. ACM J Emerg Technol Comput Syst 6(11)
19. Luo Y, Chakrabarty K, Ho T-Y (2012) Dictionary-based error recovery in cyberphysical digital-microfluidic biochips. In: proceedings of computer-aided design conference, pp 369–376

20. Hsieh Y-L, Ho T-Y, Chakrabarty K (2012) Design methodology for sample preparation on digital microfluidic biochips. In: Proceedings of international computer design conference, pp 189–194
21. Luo Y, Chakrabarty K, Ho T-Y (2012) A cyberphysical synthesis approach for error recovery in digital microfluidic biochips. In: proceedings of design, automation & test in europe conference & exhibition, pp 1239–1244
22. Alistar M, Pop P, Madsen J (2012) Online synthesis for error recovery in digital microfluidic biochips with operation variability. In: IEEE symposium on design. Test, integration and packaging of MEMS/MOEMS, pp 53–58
23. Ren H, Fair RB (2002) Micro/nano liter droplet formation and dispensing by capacitance metering and electrowetting actuation. In: proceedings of nanotechnology conference, pp 369–372
24. Srinivasan V, Pamula V, Pollack M, Fair R (2003) A digital microfluidic biosensor for multianalyte detection. In: proceedings of micro electro mechanical systems conference, pp 327–330
25. Luo Y, Chakrabarty K, Ho T-Y (2013) Error recovery in cyberphysical digital microfluidic biochips. IEEE Trans Comput-Aided Des IntegrCircuits Syst 32(1):59–72
26. Su F, Ozev S, Chakrabarty K (2004) Concurrent testing of droplet-based microfluidic systems for multiplexed biomedical assays. In: proceedings of the international test conference, pp 883–892
27. Su F, Hwang W, Mukherjee A, Chakrabarty K (2007) Testing and diagnosis of realistic defects in digital microfluidic biochips. J Electron Test 23(2–3):219–233
28. Hu K, Hsu B-N, Madison A, Chakrabarty K, Fair Richard B (2013) Fault detection, real-time error recovery, and experimental demonstration for digital microfluidic biochips. In: proceedings of the conference on design, Automation and Test in Europe, pp 559–564
29. Su F, Ozev S, Chakrabarty K (2005) Ensuring the operational health of droplet-based microelectrofluidic biosensor systems. Sens J 5(4):763–773
30. Xu T, Chakrabarty K (2007) Parallel scan-like test and multiple-defect diagnosis for digital microfluidic biochips. Trans Biomed Circuits Syst 1(2):148–158
31. Gong J, Fan S-K, Kim C-J et al (2004) Portable digital microfluidics platform with active but disposable lab-on-chip. In: Proceedings of the 17th IEEE international conference on micro electro mechanical systems (MEMS), pp 355–358
32. Su F, Chakrabarty K (2006) Benchmarks for digital microfluidic biochip design and synthesis. Duke University Department ECE, Durham
33. Luo Y, Chakrabarty K, Ho T-Y (2013) Design of cyberphysical digital microfluidic biochips under completion-time uncertainties in fluidic operations. In: ACM Proceedings of the 50th annual design automation conference, pp. 44–51
34. Yoshida J-I (2010) Flash chemistry: flow microreactor synthesis based on high-resolution reaction time control. Chem Rec 10:332–341
35. Izosimov V, Pop P, Eles P (2008) Scheduling of fault-tolerant embedded systems with soft and hard timing constraints. In: ACM proceedings of the conference on design, automation and test in Europe, pp. 915–920
36. Taylor JR (1997) An introduction error analysis: the study of uncertainties in physical measurements. University science books, Sausalito
37. Maftei E, Pop P, Madsen J (2010) Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices. Des Autom Embed Syst 14(3):287–307
38. Maftei E, Pop P, Madsen J (2009) Tabu search-based synthesis of dynamically reconfigurable digital microfludic biochips. In: Proceedings of the 2009 international conference on compilers, architecture, and synthesis for embedded systems, 195–204
39. Sinnen O (2007) Task scheduling for parallel systems. Wiley, Hoboken
40. Luo Y, Chakrabarty K, Ho T-Y (2013) Real-time error recovery in cyberphysical digital-microfluidic biochips using a compact dictionary. IEEE Trans Comput-Aided Des Integr Circuits Syst 32(12): 1839–1852