

# Performance Improvements and Congestion Reduction for Routing-based Synthesis for Digital Microfluidic Biochips

Skyler Windh, Calvin Phung, Daniel T. Grissom, Paul Pop, *Member, IEEE*, and Philip Brisk, *Member, IEEE*

**Abstract**—Routing-based synthesis for digital microfluidic biochips yields faster assay execution times compared to module-based synthesis. We show that routing-based synthesis can lead to deadlocks and livelocks in specific cases, and that dynamically detecting them and adjusting the probabilities associated with different droplet movements can alleviate the situation. We also introduce methods to improve the efficiency of wash droplet routing during routing-based synthesis, and to support non-reconfigurable modules, such as integrated heaters and detectors. We obtain increases in success rates when dealing with resource-constrained chips and reductions in average assay execution time.

**Index Terms**—Digital microfluidic biochip (DMFB), routing-based synthesis, wash droplets

## I. INTRODUCTION

SOFTWARE-PROGRAMMABLE *laboratories-on-a-chip (LoCs)* offer the potential to automate many laboratory functions that are presently performed by hand. The anticipated result is a revolution in terms of productivity and miniaturization that is poised to positively affect the biological sciences. Established applications of LoC technology include DNA sequencing, immunoassays, point-of-care diagnostics, and many others [1]. *Electrowetting-on-dielectric (EWoD)* is an emerging LoC technology that manipulates discrete droplets [2-4]. Fig. 1 illustrates the electrowetting effect: applying an electrical potential to a liquid droplet resting on a hydrophobic surface reduces the contact angle, causing the droplet to deflect. In essence, the application of an electrostatic force increases the amount of surface area that is in contact with (i.e., wetted by) the droplet; hence the name: electrowetting.

Manuscript received April 10, 2015; revised September 29, 2015 and March 12, 2016; accepted April 17, 2016. Date of publication TBD; date of current version TBD. This paper was recommended by Associate Editor T-Y. Ho. (Corresponding author: Philip Brisk.)

S. Windh, C. Phung, and P. Brisk are with the Department of Computer Science and Engineering, University of California, Riverside, Riverside, CA 92521 USA (email: {swind001, calvin.phung}@ucr.edu, philip@cs.ucr.edu).

D. Grissom is with the Dept. of Engineering and Computer Science, Azusa Pacific University, Azusa, CA 91702 USA (email: dgrissom@apu.edu).

P. Pop is with DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark (email: paupo@dtu.dk)

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

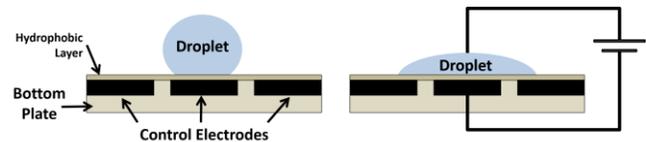


Fig. 1. Depiction of the electrowetting principle [2, Fig. 3]: applying an electrostatic potential to a droplet at rest reduces the contact angle with the surface, thereby increasing the surface area in contact with the droplet.

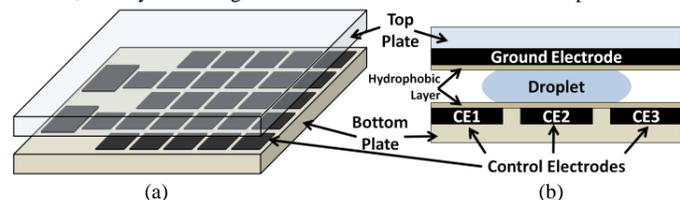


Fig. 2. (a) A DMFB is composed of a two-dimensional grid of electrodes. (b) a cross-section of a DMFB: activating CE2 holds the droplet in-place; activating CE1 and/or CE3 induces droplet motion.

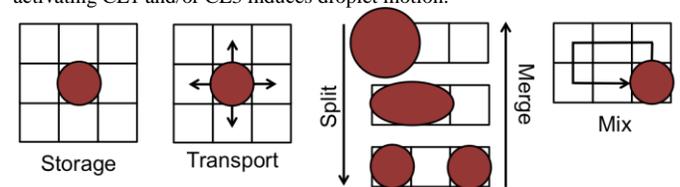


Fig. 3. The basic set of droplet operations supported by a DMFB. Other operations can be added through sensor integration and/or external devices affixed to specific regions of the chip.

Fig. 2(a) shows a software-programmable LoC based on the EWoD principle; these devices, called *Digital Microfluidic Biochips (DMFBs)*, manipulate discrete droplets of liquid on a two-dimensional grid. A DMFB comprises a two-dimensional array of individually addressable electrodes placed beneath a hydrophobic surface and a ground electrode placed atop a hydrophobic surface, with a droplet sandwiched in between, as shown in Fig. 2(b). Activating a control electrode (CE2) under a droplet holds it in-place. Activating adjacent electrode CE1 (CE3) and deactivating CE2 transports the droplet left (right).

As shown in Fig. 3, the DMFB instruction set includes droplet transport in two dimensions, splitting, merging two droplets into one, mixing, and storage in-place. Additionally, external devices such as heaters [5], photo-detectors [6, 7], capacitance sensors [8], impedance sensors [9], or magnetic separators [10] can be affixed to specific regions of the DMFB to offer additional functionality: to use one of these external devices, a droplet is transported to an appropriate location on-chip and then stored in-place while the operation is performed.

There has been interest in the development of programming languages and compiler technology targeting DMFBs in recent years [11-41]. As shown in Fig. 4, a “program” that executes a biological protocol on a DMFB is a sequence of electrode activations that execute the protocol one step at a time. The activation sequence can be viewed as a linear state machine (a Moore machine) in which the output of each state is a bit-vector, where each ‘1’ represents an electrode that is activated and each ‘0’ represents an electrode that is not.

Most compilation work targeting DMFBs assumes that the execution of operations is constrained to a group of adjacent electrodes forming a rectangular called “module;” however, the reconfigurable operations (e.g., mixing, dilution) can execute by routing the droplets on any sequence of electrodes on the microfluidic array [11]. One drawback of rectangular modules is that the droplets will only occupy a subset of the rectangular region dedicated to the operation at any given time, which yields poor utilization of spatial resources. A second drawback is that module selection and placement are inherently tied to the notion of rectangular mixing modules.

*Routing-based synthesis* (Fig. 5) is an alternative to module-based compilation [12]. Routing-based synthesis eliminates the concept of “modules” and allows the droplets to move on the chip on any route during operation execution. Routing-based synthesis converts concurrent mixing operations into a routing problem: mixing droplets can move anywhere on the DMFB as long as they do not inadvertently interfere.

**Contribution:** Routing-based synthesis, as described in Ref. [12], supports mixing and dilution operations, but not droplet storage and/or operations that rely on external devices, such as heating or detection. Randomly generating droplet movements during routing-based synthesis leads to livelock and deadlock situations. We introduce new approaches to droplet movement generation that significantly reduce the likelihood that these catastrophic situations occur. We integrate routing-based synthesis with a known effective and efficient scheduling heuristic, and introduce modifications to compensate for the fact that operation completion times are not known statically and are instead determined by the sequence of droplet movements generated by the algorithm. Lastly, we reduce washing overhead by shortening the length of the paths that wash droplets must travel to perform local decontamination. The result of this effort is an enhanced routing-based synthesis implementation that supports all known protocol operations and is less susceptible to failures than the original.

## II. RELATED WORK

Figs. 6 illustrates a DMFB compiler. The input is an assay as a *directed acyclic graph (DAG)*: vertices represent operations (e.g., input/dispense, mix, detect, etc.), and edges represent dependencies between operations, i.e., an edge  $(x, y)$  indicates that operation  $x$  produces a droplet that will later be consumed by operation  $y$ . The compiler must solve three interdependent NP-complete problems to produce an executable program to control the DMFB. The compiler assumes that capacitance sensors or real-time video monitors detect droplet presence and that droplets are carried in a filler fluid (e.g., silicone oil) to prevent evaporation.

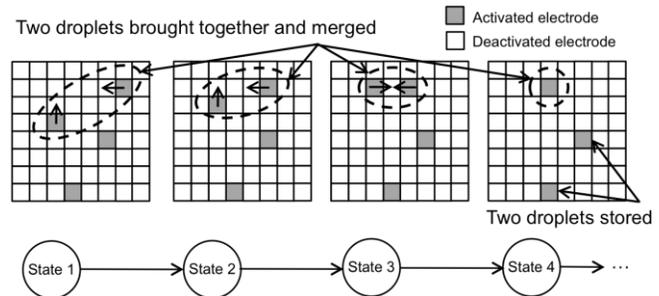


Fig. 4. The output of a DMFB compiler is a linear state machine that outputs an electrode activation sequence to execute the protocol. This state machine should not be confused with the DAG representation of the protocol (Fig. 6).

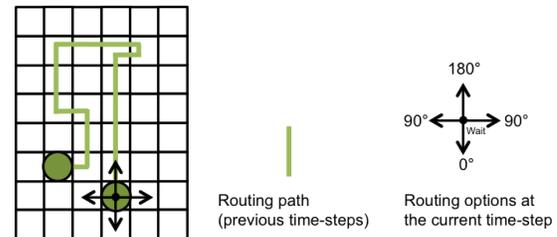


Fig. 5. Illustration of routing-based synthesis for a single droplet.

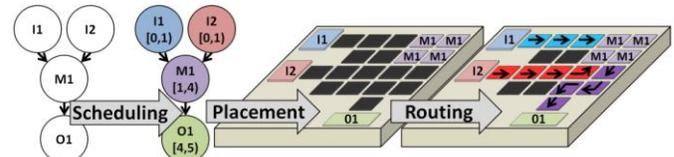


Fig. 6. Illustration of a typical compiler targeting a DMFB: the protocol to execute is represented as a DAG. The compiler must schedule, place, and route all operations in order to produce an executable sequence of electrode activations to automatically run the protocol.

### A. Scheduling

The compiler must determine the time at which each operation starts and finishes [13-18]. All droplet dependency constraints, as specified by DAG edges, must be satisfied, i.e., for edge  $(x, y)$ , operation  $x$  must finish before  $y$  begins. The number of operations scheduled to execute at any given time cannot exceed the resource capacity of the DMFB (Fig. 6). Most schedulers assume that all mixing operations use modules of the same size; it can also be integrated with a separate *module selection step* [19-22] mixing and dilution are performed by bringing two (or more) droplets together and rotating them according to a given pattern [11]. The operation completion time varies, depending on the size of the mixer. In general, larger mixers yield shorter operation times, but consume more on-chip resources, limiting the amount of parallelism available to perform other operations concurrently.

Existing work in scheduling includes fast, greedy heuristics [13-16], genetic and metaheuristic-based heuristics [13, 17], and optimal algorithms based on integer linear programming (ILP), which runs in exponential worst-case time for specific application domains (PCR [18] and in-vitro diagnostics [13]). The scheduler employed by our implementation of routing-based synthesis is similar to Path Scheduling [14], which performs well when spatial resources are limited.

Although many DMFB compiler algorithms include module selection, it has not been treated as a standalone problem. For example, several iterative improvement algorithms randomly vary the module assignment for each protocol operation, but

solve it in conjunction with scheduling and placement [19-22]. Another algorithm performs module selection as a post-processing phase after scheduling (to reduce latency), but fails to account for area constraints of the target chip [17]. Routing-based synthesis sidesteps module selection by converting executable operations into a routing problem.

### B. Placement

If operation  $x$  is scheduled to start at time  $t_1$  and finish at time  $t_2$ , a location on the surface of the chip must be reserved for  $x$  during the time interval; no other operation may occupy the same location during this time interval in order to prevent accidental merging and cross-contamination of concurrent operations [23-32] (Fig. 6). DMFBs are spatially parallel and reconfigurable, as the roles played by individual electrodes vary over the execution of a protocol (e.g., transport, storage, mixing, etc.). An effective placer must represent free space on the chip, and allocate and deallocate space as operations start and finish; greedy [23, 24], iterative improvement [25], and optimal [26] placement algorithms have been proposed. Virtual topologies [27-30] partition a DMFB into regions that perform operations with dedicated routing channels between them; this converts placement into a simpler binding problem.

In contrast, routing-based synthesis converts placement into a routing problem. Combined scheduling and placement can be modeled as a 3D placement problem, (the third dimension is time) [31, 32]. This approach is incompatible with routing-based synthesis, which does not employ any notion of rectangular (2D) or cuboid (3D) modules.

### C. Routing

When droplets are produced/consumed by operations, they are transported from one location on the chip to another, in accordance with the schedule and placement results [33-38]. During transport, droplets must maintain appropriate spacing (Fig. 7) and may not inadvertently intersect regions of the chip performing mixing and storage operations [33]. Routing may be integrated with *washing* [39-41] to clean residue left by other droplets and completed operations (Fig. 8).

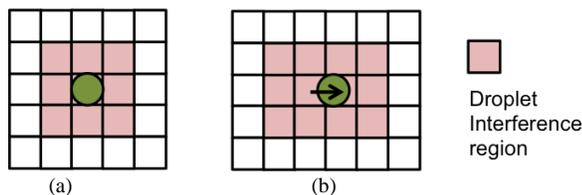


Fig. 7. The interference region for a droplet at rest (a) and during transport (b). If any droplet enters the interference region of another, then they will merge inadvertently [33].

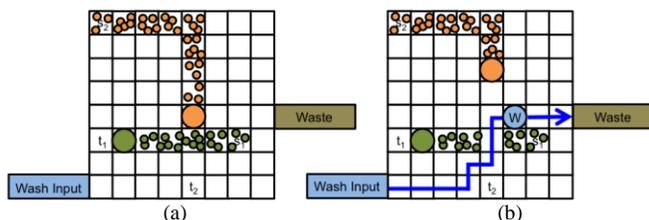


Fig. 8. Illustration of cross-contamination: (a) droplets leave residue behind when traveling across the surface of the chip; (b) when two droplet routes intersect, a wash droplet (W) must clean the intersection point before the second droplet can proceed.

To reduce cross-contamination, these algorithms try to route droplets along disjoint paths; wash droplets are only introduced when disjoint paths cannot be found. In contrast, Routing-based synthesis moves droplets randomly without pre-defined starting and ending points; one contribution of this work is to reduce the likelihood that deadlock or livelock occurs; our implementation also incorporates wash droplets.

## III. ROUTING-BASED SYNTHESIS

This section summarizes routing-based synthesis as described by Maftai et al. [12]; the limitations of their approach are described, setting the stage for our corrections, which are presented in Section IV.

### A. Rectangular Mixing Modules

Paik et al. [11] studied the mixing times of rectangular modules of varying dimensions, as reported in Table I; these results assume a 1.5 mm electrode pitch, 600  $\mu\text{m}$  gap height, 16 Hz switching frequency, and 1.4  $\mu\text{L}$  droplets for mixing.

Linear array mixers ( $1 \times N$ ) move a merged droplet in one dimension (e.g., left/right). Reversing direction causes flow reversibility, which works against effective mixing; the ratio of forward to reversing movements is  $R = (N-2)/(N-1)$ , e.g., 2/3 for a  $1 \times 4$  mixer. Increasing  $N$  increases  $R$ , but at the expense of consuming more on-chip area.

In a  $2 \times 2$  mixer, the merged droplet is mixed via rotation about a pivot; all rotations follow one angular direction (clockwise/counterclockwise); the angular direction does not reverse. Although the  $2 \times 2$  mixer eliminates flow reversal effects, a portion of the droplet near the pivot mixes slower than the rest of the droplet, which slows overall mixing times.

The  $2 \times 3$  mixer eliminates the static pivot, and includes forward motions in addition to (counter-)clockwise turns. It offers a significant improvement over the  $2 \times 2$  mixer, and eliminates the flow reversibility of the  $1 \times 4$  mixer, although it is still slower than the  $1 \times 4$  mixer. This suggests that increase the ratio of forward movements to both turns and reversals will have the greatest possible effect on total mixing time.

The  $2 \times 4$  mixer combines the benefits of the  $1 \times 4$  and  $2 \times 3$  mixers, yielding the best overall mixing time. With a  $2 \times 4$  mixer, a droplet may take many different paths, as shown in Fig. 9(a) and (b). In effect, the routing paths within the modules have been optimized via routing based synthesis.

TABLE I  
TIMES FOR DMFB OPERATIONS INCLUDING MIXING/DILUTIONS USING VARYING RECTANGULAR DIMENSIONS

Operation	Area	Time (s)
Mix/Dilute	2x2	9.95
Mix/Dilute	2x3	6.1
Mix/Dilute	1x4	4.6
Mix/Dilute	2x4	2.9
Dispense	----	2
Detect	1x1	30

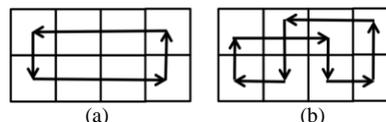


Fig. 9. Two mixing paths within a  $2 \times 4$  array mixer that yield optimal mixing times of 2.9s as reported by Paik et al. [11].

### B. Routing-based Mixing

Maftei et al. [12] examined the results of rectangular mixing modules [11] and determined the percentage of mixing achieved by decomposing the routes within the modules into basic movements:  $p^0$  denotes the percentage of mixing obtained by moving a droplet forward;  $p^{90}$  denotes the percentage of mixing obtained by turning a droplet left or right; and  $p^{180}$  denotes the percentage obtained by reversing the direction of the droplet. Maftei et al. further decomposed  $p^0$  into two separate values,  $p_1^0$  when the forward move is one cell, and  $p_2^0$  when the forward movement is two or more cells.

Let  $\mu = \{p_1^0, p_2^0, p^{90}, p^{180}\}$  be the set of percentages; Maftei et al. [12] empirically determined the following values for  $\mu$ :  $p_1^0 = 0.29\%$ ,  $p_2^0 = 0.58\%$ ,  $p^{90} = 0.1\%$ , and  $p^{180} = -0.5\%$ , which accounts for negative mixing affects due to flow reversibility. Any sequence of droplet movements that adds up to 100% can mix two droplets; there is no requirement to constrain these movements to a rectangular sub-region of the DMFB.

### C. Application Model

An assay is represented as a DAG  $G = (V, E)$ , as shown in Fig. 4. Each vertex  $v_i \in V$  is an operation, e.g., mix, dilute, split, detect, dispense, output, etc. Edge  $(v_i, v_j) \in E$  is a dependency, i.e., operation  $v_i$  produces a droplet that is used by subsequent operation  $v_j$ ;  $v_j$  must wait for all constituent droplets to arrive before it can start execution. A droplet not used immediately after it is produced (per the schedule) must be stored on-chip, consuming spatial area that would be otherwise be allocated to operations that drive the protocol toward completion [14].

### D. Routing-based Synthesis Algorithm

Fig. 10 shows pseudocode for routing-based synthesis [12], which is limited to reconfigurable operations (e.g., mixing) and wash droplet transport. Non-reconfigurable operations, which are not supported, include fluid I/O and the usage of external devices (e.g., heaters, detectors, etc.), where a droplet must be routed to a specific on-chip region and held in-place; the latter is mentioned in passing but the algorithms to support it are not discussed. Likewise, production of waste droplets and routing droplets off-chip is not addressed.

The inputs to the routing-based synthesis are the DAG representation of the protocol to execute ( $G$ ), an array representing the DMFB ( $C$ ), the set of mixing percentages ( $\mu$ ), and two parameters representing properties of wash droplets ( $max_{electrodes}$  and  $no_{part}$ ). Each wash droplet has a finite capacity for contamination removal [12]: it can clean  $max_{electrodes}$  cells of the DMFB, after which it must be discarded and replaced with a new wash droplet. Routing-based synthesis, as described by Maftei et al., partitions the DMFB into a set of  $no_{part}$  distinct regions: one wash droplet is allocated to each region, and that wash droplet removes all contamination within its region and that region alone; when its capacity is exceeded, it is replaced with a new wash droplet.

A DMFB operates at 100Hz, meaning that it takes 10ms to move a droplet from one electrode to a neighbor. Assuming fully synchronized droplet movements at equal velocities, we refer to each 10ms interval as a *time-step*. The algorithm starts at time-step 0. Variable  $t_{current}$  tracks the current time-step. For each protocol operation  $v_i$ ,  $t_i^{start}$  and  $t_i^{finish}$  represent the start and end times, as computed by the algorithm.

RoutingBasedSynthesis( $G = (V, E), C, \mu, max_{electrodes}, no_{part}$ )

```

1.  $t_{current} = 0$ 
2. for each operation  $v_i \in V$  do
3.    $t_i^{start} = t_i^{finish} = 0$ 
4. end for
5.  $L_{merge} = \text{ConstructMergeList}(G)$ 
6.  $L_{mix} = \emptyset$ 
7.  $L_{wash} = \text{ConstructWashList}(max_{electrodes}, no_{part})$ 
8.  $\text{PartitionChip}(C, no_{part})$ 
9. while  $\exists v_i \in V \wedge t_i^{finish} = 0$ 
10.  for all  $v_i \in L_{merge} \cup L_{mix}$  do
11.     $R_i = \text{PerformMove}(v_i, C, \mu, R, t_{current})$ 
12.    if  $v_i$  causes contamination then
13.       $\text{SetElectrodeContaminated}(v_i, R_i, L_{wash})$ 
14.    end if
15.  end for
16.  for all  $v_i \in L_{merge}$  such that  $v_i$  is merged do
17.     $\text{Remove}(v_i, L_{merge})$ 
18.     $\text{ScheduleSuccessors}(v_i, L_{mix})$ 
19.  end for
20.  for all  $v_i \in L_{mix}$  such that  $v_i$  has completed mixing do
21.     $t_i^{finish} = t_{current}$ 
22.     $\text{Remove}(v_i, L_{mix})$ 
23.    if  $v_i$  has successor operations then
24.       $\text{ScheduleSuccessors}(v_i, L_{merge})$ 
25.    end if
26.  end for
27.  for all  $v_i \in L_{wash}$  do
28.     $R_i = \text{PerformMove}(v_i, C, \mu, R, t_{current})$ 
29.    if  $R_i$  is contaminated then
30.       $\text{SetElectrodeCleaned}(R_i, L_{wash})$ 
31.       $\text{UpdateWashCapabilities}(v_i)$ 
32.      if  $v_i$  has exhausted its washing capacity then
33.         $\text{RemoveFromWashList}(v_i, L_{wash})$ 
34.         $\text{CreateNewWashDroplet}(v_i, L_{wash})$ 
35.         $\text{SetRouteTarget}(v_i, \text{Waste})$ 
36.      end if
37.    end if
38.  end for
39.   $t_{current} = t_{current} + 1$ 
40. end while
41. Return  $R$  // set of routes

```

Fig. 10. Pseudocode for routing-based synthesis [12]. Operations that support cross-contamination removal are shown in blue.

Considering only reconfigurable operations, active droplets are either the *merge* or *mix* state. An active droplet is in the merge state if it is ready to mix with another droplet, but the two have not yet merged; it is in the mix state during mixing. Lists  $L_{merge}$  and  $L_{mix}$  denote the sets of active droplets in these states respectively;  $L_{wash}$  is the set of active wash droplets.

**Lines 1-8:** The first eight lines initialize the algorithm.  $L_{merge}$  contains all droplets that can be dispensed immediately,  $L_{mix}$  is empty, and  $L_{wash}$  contains one droplet per partition (Line 8).

**Lines 9-40:** The algorithm proceeds until all droplets are processed, indicated by a non-zero completion time; droplets are processed in topological order.

**Lines 10-15:** This for all loop moves all assay droplets in lists  $L_{merge}$  and  $L_{mix}$ . Each droplet has at most 5 possible moves: {*up, down, left, right, hold*}. In the case of a *hold*, the droplet does not move. The algorithm considers only legal move operations. If a droplet is at the perimeter of the chip, it is not allowed to move off of the chip; a droplet cannot make a move that causes it to inadvertently merge with another droplet, i.e., droplets have to observe the well-established spacing rules [33] (Fig. 7); lastly, a droplet cannot move onto a region of the chip that has been contaminated by residue left by another droplet.

All legal moves are ranked in terms of their profitability. For a droplet  $d$  in  $L_{merge}$ , profitability is determined based on whether a given move is toward, neutral, or away from the droplet  $d'$  with which  $d$  is supposed to merge. For droplets in  $L_{mix}$  profitability is determined by the mixing percentages stored in set  $\mu$  (the mixing percentage of a *hold* is 0%). The most profitable move is randomly selected with a probability of 50%, the second most profitable move is randomly selected with a probability of 33.3%, and the third most profitable move is randomly selected with a probability of 16.7% (Line 11). Any time a droplet moves onto a new cell, it contaminates that cell with residue (Lines 12-14).

**Lines 16-19:** When two droplets in  $L_{merge}$  merge, they form a single droplet, represented by a successor vertex in the DAG. The droplets are removed from  $L_{merge}$  and the successor is added to  $L_{mix}$ ; mixing commences during the next time-step.

**Lines 20-26:** If a droplet in  $L_{mix}$  completes its mixing operation, then it finishes its operation at the current time-step (Line 22), and is removed from  $L_{mix}$  (Line 23). Any successor operations may commence if all of their predecessors have finished; if so, they are added to the set  $L_{merge}$  (Lines 23-25).

**Lines 27-38:** Wash droplets are moved randomly as well; profitability is computed based on the Manhattan distance between the current position of the wash droplet and the first electrode to be cleaned (Line 28). If the wash droplet moves onto a contaminated cell (Lines 29-37), then the cell is updated to reflect the fact that it has been decontaminated (Line 30) and the wash capacity of the wash droplet is reduced by one (Line 31). If the wash droplet's wash capacity is reduced to zero (Lines 32-35), then the wash droplet is removed from  $L_{wash}$  (Line 33), a new wash droplet is dispensed to take its place (Line 34), and the original wash droplet is set on a path to a waste reservoir (Line 35). Maftei et al. do not describe precisely how the wash droplet travels to the waste reservoir; we assume that the mechanism is similar in principle to the way droplets in  $L_{merge}$  are handled.

One final implementation option, not discussed above and omitted from the pseudocode, is to partition the chip [12] so that each mixing operation occurs in a different partition; mixing droplets move about randomly within their partitions, but cannot cross the partition boundary. Individual droplets may cross partition boundaries to merge, and/or to leave the chip. This modification prevents cross-contamination between mixing operations. Depending on the size and dimensions of each partition, the scheme may degenerate into a traditional rectangular module-based synthesis scheme.

#### IV. IMPROVEMENTS TO ROUTING-BASED SYNTHESIS

This section highlights several limitations of routing-based synthesis as described by Maftei et al. [12], as well as a set of practical improvements that overcome these limitations.

##### A. I/O Reservoir Blockage

We have observed specific situations where droplet I/O operations can lead to deadlocks that cannot be reconciled. The solution to this problem is to prevent other droplets from entering a small set of cells around each port, so that droplets can always have the opportunity to enter/exit the chip.

**Input Reservoir Blockage:** Contaminated cells near an input reservoir can cause an unfixable deadlock. In Fig. 11(a), the green droplet has contaminated all cells next to an input reservoir, except for the cell into which a droplet is dispensed. In Fig. 11(b), the dispensed droplet cannot move because all adjacent cells are contaminated (we assume that the droplet cannot be un-dispensed back into the input reservoir). Cleaning any of the contaminated adjacent cells would inadvertently merge the wash droplet with the green droplet.

Fig. 11(c) depicts a solution. Before dispensing a droplet, all cells within the 2x3 region adjacent to the reservoir must be contamination-free and contain no other droplets. If so, the dispense operation may proceed; otherwise, it is delayed until the aforementioned criteria are satisfied. This situation only occurs with respect to input reservoirs dispensing droplets. It is impossible for one droplet to trap another that is already on the chip in the same manner, as contaminating the adjacent cells would merge them in violation of spacing rules [33].

**Wash Droplet Input Reservoir Blockage:** Fig. 12(a) illustrates a situation in which contamination left by a green droplet traps an orange droplet in the 2x3 region adjacent to a wash droplet reservoir. The input reservoir cannot dispense the wash droplet because it would inadvertently merge with the orange droplet at any position in the 2x3 region. The problem here arises due to the finite capacity of wash droplets. Unlike Fig. 11, the location of the orange droplet does not prevent wash droplets from cleaning cells that the green droplet contaminated. If all wash droplets on the chip have infinite washing capacity, then eventually all cells will be cleaned; however, with finite capacities, all wash droplets on may reach their respective capacities before they can clean these cells. If a wash droplet is at its capacity, it is routed to a waste reservoir for disposal. If all wash droplets are removed, then the assay deadlocks, as it is impossible to inject another wash droplet because the input reservoir is blocked.

One solution is to employ multiple wash droplet dispense reservoirs; however, the possibility remains that all reservoirs could be simultaneously blocked under the same scenario (although the probability of this goes down with each additional wash droplet reservoir). A second solution is to move the orange droplet out of blocked region, contaminating it; the orange droplet is no longer usable and must be sent to a waste reservoir for disposal. An error recovery procedure can then be invoked to re-generate the orange droplet [7]; however, this incurs non-negligible performance overhead and would lengthen the execution time of the assay significantly.

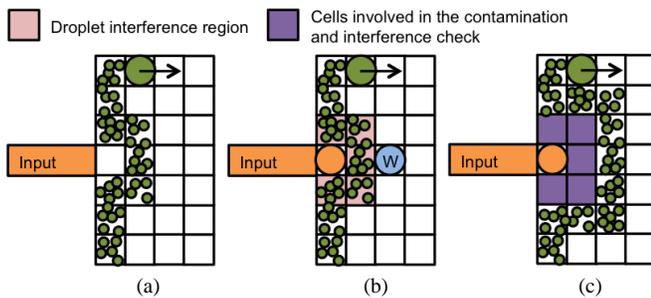


Fig. 11. (a) The cell adjacent to the input reservoir is not contaminated, so the input reservoir can safely dispense a droplet. (b) The orange droplet cannot move since all neighboring cells are contaminated. The wash droplet  $W$  cannot clean the contaminated cells adjacent to the orange droplet without inadvertently merging the two droplets. (c) To rectify the situation, a droplet can only be dispensed if all cells in the  $2 \times 3$  region adjacent to the input reservoir are free of contamination and do not contain any other droplets.

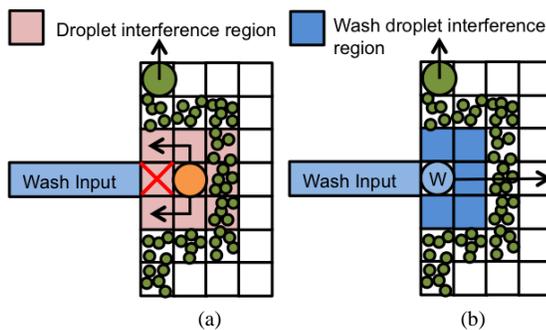


Fig. 12. (a) A droplet is trapped in a  $2 \times 3$  region adjacent to a wash droplet input reservoir, blocking it. (b) The solution is to immediately inject the next wash droplet  $W$  and store it next to the input reservoir; this ensures that no other droplets can enter the  $2 \times 3$  region, blocking the wash droplet input reservoir. If another wash droplet reaches its capacity,  $W$  is transported away from the input reservoir to replace it; a new droplet can be dispensed immediately, ensuring that no other droplet enters the  $2 \times 3$  region.

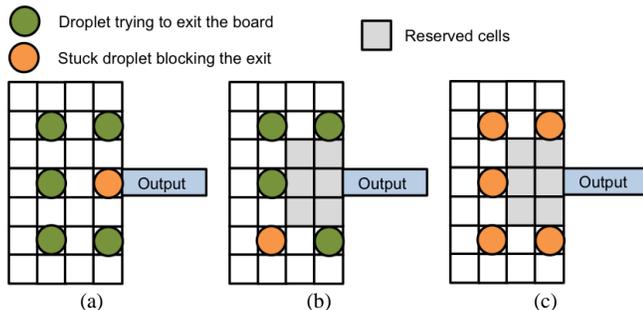


Fig. 13. (a) The orange droplet blocks the green droplets that are trying to exit the chip; the green droplets block the orange droplet's path out of the way, while the orange droplet blocks all of the green droplets' paths to the output reservoir. (b) Reserving a  $2 \times 3$  region adjacent to the output reservoir ensures that one orange droplet cannot cause a blockage; (c) however, multiple orange droplets can still block access to the output reservoir.

Fig. 12(b) shows our solution. Each wash droplet reservoir immediately dispenses a new wash droplet and holds it in-place until it is needed. When a wash droplet is needed, it is transported to its decontamination region, and the next wash droplet is immediately dispensed and held. This eliminates the deadlock situation shown in Fig. 12(a): contamination could still trap an assay droplet in a larger region that subsumes the wash droplet entry point, however, that droplet would be trapped *with* an otherwise unused wash droplet, which ensures that the contamination can and will be removed eventually.

**Output Reservoir Blockage:** Fig. 13(a) illustrates a situation where a one droplet (orange) blocks several droplets (green) trying to exit the chip. This situation is difficult to detect and rectify with 100% certainty, because it can scale up to an arbitrary number of droplets in the most general case. In Fig. 13(b), we reserve a  $2 \times 3$  region adjacent to the output reservoir; the only droplets that may enter the region are those in the process of exiting the chip; thus, one orange droplet cannot block a set of green droplets that want to exit; however, as shown in Fig. 13(c), a group of 5 orange droplets could still cause a similar blockage, although the likelihood of this situation occurring is much lower than the single-droplet blockage in Fig. 13(a). Further techniques to rectify this situation will be discussed in the following subsections.

### B. Support for Non-Reconfigurable Operations

Maftai et al. [12] categorize I/O and the usage of external devices (e.g., heaters and detectors) as *non-reconfigurable* operations, because they must occur at specific locations on the DMFB; transport, mixing, splitting, and merging can occur anywhere on the device—hence, they are *reconfigurable*. Maftai et al. mention that non-reconfigurable operations involve routing a droplet to a specific (subset of) location(s) on the DMFB, which can be handled probabilistically (e.g., a move toward the target is favorable); however, we have uncovered several issues left unaddressed by their work.

Without loss of generality, suppose that routing-based synthesis wants to use a heater  $H$  to increase the temperature of droplet  $d$ .  $H$  is an  $m \times n$  sub-region of the DMFB. Two things must happen: (1) all droplets other than  $d$  must be routed out of  $H$  and may not return into  $H$  until the heating operation completes; and (2)  $d$  must be routed onto  $H$ . Maftai et al.'s description addresses the second requirement, but not the first. Here, we consider droplet output, and operations that use external modules at pre-specified locations on the chip.

**Output Operations:** A droplet  $d$  is removed from the DMFB by routing it to an output reservoir for collection or a waste reservoir for disposal. Suppose that  $d$  becomes ready for output when it holds a position at DMFB location  $(x, y)$ .

To route  $d$  to the output reservoir, there are two options: (1) pre-compute a routing path from  $(x, y)$  to the output reservoir, similar to traditional DMFB routing algorithms [12], and move  $d$  along this path (pausing to prevent interference with other droplets); or (2) transport  $d$  from  $(x, y)$  to the output reservoir probabilistically using routing-based synthesis, similar in principle to the way that droplets in  $L_{merge}$  are handled in Fig. 10. A move is profitable if it transports the droplet toward the output reservoir, unprofitable if it transports the droplet away, and neutral if the droplet holds its position.

We chose the latter option because: (1), it allows us to maintain uniformity with the routing procedure applied to other droplets on the chip; and (2) it allows us to back out of the output blockage situation shown in Fig. 13(a). Although unlikely, one or more green droplets could back away from the congested region by the output reservoir, providing a path for the orange droplet to leave the area, freeing up space for the green droplets exit the chip. We can also selectively increase the probability to accept unprofitable moves, increasing the likelihood of recovering from output reservoir deadlocks.

**External Modules:** In principle, mixing or dilution can occur anywhere on the DMFB, including on top of an external device, e.g., a heater, when not in use; however, at some point, a droplet  $d$  may need to use the external device. Then (1) all droplets other than  $d$  must be moved away from the region affected by the device; and (2)  $d$  must be routed to the region.

We refer to an  $m \times n$  sub-region of a DMFB that is affected by an external device as a *module*, denoted by  $M$ . One solution is to lock  $M$  so that only droplets bound to operations that use the external device associated with  $M$ ; the drawback is that doing so reduces the available on-chip area when external devices are not used, which limits parallelism. The alternative is to leave  $M$  open for use by reconfigurable operations when the external device is not in use, and then evict all ongoing operations from  $M$  when the external device is used. We have taken this approach in our implementation of routing-based synthesis; we describe the eviction process in detail here.

When no non-reconfigurable operations execute on  $M$  droplets undergoing mixing and transport may enter and exit  $M$  without restriction. When a non-reconfigurable operation  $v_i$  executes on  $M$  all droplets not bound to  $M$  and currently residing on  $M$  must be moved out; no droplets other than those needed for operation  $v_i$  may enter  $M$  until  $v_i$  completes.

We tried two schemes to remove droplets from  $M$ . The first forces droplets to move directly out of  $M$ , temporarily bypassing the randomized process of routing-based synthesis. We observed empirically that this could lead to deadlocks if there was high congestion surrounding  $M$ . The second scheme employs randomization to remove unwanted droplets from  $M$ . Each cell  $c_j$  within  $M$  is assigned a positive integer value called the *depth* of  $c_j$ , which represents the distance from  $c_j$  to the perimeter of  $M$ , as shown in Fig. 14(a). For each unwanted droplet, we temporarily change the objective from minimizing operation completion time to routing away from  $M$ ; once the droplet is removed, the objective then reverts. A droplet that exits  $M$  cannot re-enter  $M$  until the non-reconfigurable operation completes, avoiding deadlocks while clearing  $M$ .

During the escape process, a favorable move is one that moves a droplet within  $M$  from a cell having higher depth to one having lower depth (i.e., toward the perimeter); moving a droplet from one cell to a neighbor having the same depth is a neutral move; negative moves, i.e., moving a droplet from a lower-depth cell to a higher depth cell (i.e., toward the center) are not permitted. Fig. 14(b) shows an example: one positive move is followed by four neutral moves, and then two positive moves, after which, the droplet fully escapes from  $M$ .

Similar to the case of output operations, any droplet that will use the external device associated with  $M$  is transported probabilistically to  $M$  using routing-based synthesis; the droplet may enter  $M$  at any time, even while other droplets are still escaping from  $M$ . Probabilistic transport maintains uniformity with the routing procedure as applied to other droplets, and prevents deadlock from occurring.

### C. Dynamic Adjustment of Droplet Movement Probabilities

The scenario illustrated in Fig. 13(a) can be viewed as either a deadlock or a livelock, depending on how routing is implemented. If the green droplets exiting the board travel along pre-computed paths without the ability to backtrack, then the situation is a deadlock, as no droplet can move.

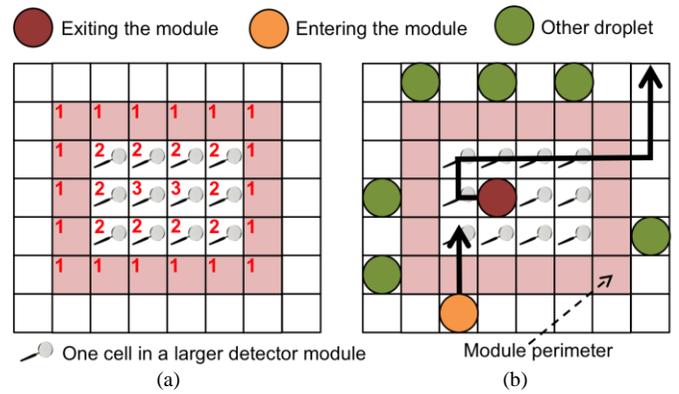


Fig. 14. (a) The assignment of depth values to a detection module  $M$ . (b) The red droplet randomly exits the detection module, while the orange droplet enters the module for detection. Note that the maroon droplet traverses the module counter-clockwise at depth level 2, before discovering an exit point.

If the green droplets travel probabilistically according to the principles of routing-based synthesis, then the situation is more of a livelock: probabilistically, a droplet may accept an unprofitable move away from the congested region; however, the following move will most likely be profitable (toward the exit), thereby restoring the congested state. This is a livelock because droplets can move, but the congestion is not resolved.

Our solution is to dynamically estimate localized congestion on the DMFB and to use this information to adjust the probabilities of accepting profitable, neutral, and unprofitable moves. The estimate must be computationally efficient and should not significantly increase assay execution time.

In Maftai et al.'s implementation of routing-based synthesis, all droplets in lists  $L_{mix}$  and  $L_{merge}$  are moved randomly. All legal moves are enumerated, the profitability of each move is estimated, and the top three legal moves in terms of profitability are ranked in-order; one of the top three is then chosen randomly with probabilities  $p = (50\%, 33.3\%, 16.7\%)$ .

Statically changing the probability values was unsuccessful. Empirically, we observed that increasing the probability that the most profitable movement is chosen tends to reduce the completion time of protocol operations when there is ample space on the chip, but stalls progress when the board becomes congested. The best way to reduce congestion is for many droplets on the perimeter of the congested area to back away; however, reversing direction is unprofitable because of unmixing [11, 12]. Meanwhile, increasing the likelihood of selecting a less profitable move tends to reduce congestion, but does not favor fast protocol completion times.

We achieved a favorable tradeoff between performance (protocol execution time) and congestion avoidance through dynamically adjusting the priority scheme by which moves are selected. We consider three probability sets as follows:

$$\begin{aligned}
 p_A &= (85\%, 10\%, 5\%), & (2) \\
 p_B &= (50\%, 33\%, 17\%), \text{ and} & (3) \\
 p_C &= (34\%, 33\%, 33\%). & (4)
 \end{aligned}$$

Our first approach was to adjust the probabilities based on the number of droplets on the board at a given time; increasing the number of droplets tends to increase the likelihood of congestion, deadlocks/livelocks being the most extreme form.

The problem with this approach is that a relatively small number of droplets can form a localized deadlock/livelock, as is the case in Fig. 13(a). It is computationally inefficient to enumerate different sub-regions of the chip and/or subsets of droplets to estimate local congestion. Instead, we turned to a more effective approach based on operation completion times.

**Operation-driven Profitability:** This scheme assigns the probability set based on an estimate of the number of time-steps required to complete each operation. For example, if droplets  $d_1$  and  $d_2$  need to merge, then half of the Manhattan Distance between them is a reasonable estimate of the number of movements required to merge the two droplets. For an ongoing mixing operation, we can estimate the number of movements required to complete it based on an expected distribution of droplet movements, each of which contributes (positively or negatively) to the total mixing time.

A droplet that is effectively stuck due to a deadlock or livelock will not advance toward completion of its operation; thus, we can use the disparity between the expected completion time and the progress obtained thus far as a proxy for deadlock or livelock detection. If we suspect that a droplet is deadlocked or livelocked, we increase the randomness in the probability set to increase the likelihood of mitigating the deadlock or livelock; otherwise, we favor movements that lead to the fastest possible completion time for the operation.

The scheme is implemented as follows: let  $O_i$  be the operation in question,  $t_{current}$  be the current time-step,  $t_i^{start}$  be the time at which operation  $O_i$  started ( $t_i^{start} \leq t_{current}$ ), and  $t_i^{predicted}$  be the estimated length to complete operation  $O_i$ . As there will be some natural variation in operation completion time due to the random selection of moves, many operations will complete after  $t_i^{predicted}$ . The longer any given operation takes to complete, the more severe we assume that the congestion must be. In response, we dynamically between the probability sets,  $p_A$ ,  $p_B$  and  $p_C$  to increase the likelihood that the droplet successfully moves out of the congested area.

We estimate that congestion occurs when the latency of an ongoing operation exceeds twice its predicted latency:

```

if  $t_{current} \leq t_i^{start} + 2 * t_i^{predicted}$ 
     $p = p_A$ 
else if  $t_{current} \leq t_i^{start} + 3 * t_i^{predicted}$ 
     $p = p_B$ 
else
     $p = p_C$ 
end if
    
```

A smaller constant value could enable more rapid transitions between probability sets, but could also lead to more false positives and increased assay execution times.

#### D. Congestion-Aware Scheduling

The pseudocode in Fig. 10 dispatches each assay operation as soon as all of its predecessors complete (except, presumably, dispensing operations) [12]. This approach may be ineffective when targeting small chips with limited spatial resources. The introduction of wash droplets, which perform a useful and necessary function, also reduce the availability of spatial resources for assay operations.

To mitigate congestion due to resource limits, we modified the operation dispatcher (Fig. 10, Line 24) to consider the impact of each operation on resource demand at future points in the schedule. Motivated by Path Scheduling [14], which was shown to be effective when targeting resource-constrained DMFBs [14, 15], we delay the execution of assay operations whose completion increases demand for spatial resources.

The *independent path priority (IPP)* of a DAG vertex  $v$ , denoted  $ipp(v)$ , is the number of leaf vertices reachable from  $v$ ; Fig. 15 shows two examples, where each vertex is labeled with its IPP value [14, Fig. 6]. The IPP estimates the increase in resource demand that will occur as a result of executing the operation. Let  $R$  be represent a DMFB's available spatial resources, let  $U$  be a set of executing operations,  $S$  be a set of operations that are presently stored (presumably consuming 1 spatial resource) and let  $IPP(U) = \sum_{u \in U} ipp(u)$ .

The scheduler dispatches a ready-to-execute operation  $v$  if  $ipp(v) + IPP(U) + |S| \leq R$ ; in other words,  $v$  may execute if the scheduler believes that present and future demands for spatial resources from all ongoing operations, plus  $v$ , are within the chip's capacity. Otherwise,  $v$  is delayed until more resources are available. Ready-to-execute operations are processed in increasing order of IPP value, which favors the dispatch of operations whose fanout trees have low resource demands; execution of those fanout trees clears space on the DMFB to execute operations with higher resource demands.

This reduces congestion on-chip, which has two favorable benefits: (1) it reduces the likelihood of global deadlocks and livelocks; and (2) alleviating congestion increases the likelihood that ongoing operations select profitable, rather than neutral or unprofitable, moves, thereby favoring faster assay execution times. With respect to Fig. 15(a), and similar to Path Scheduler [14], this approach favors continuing execution along existing paths that have already been started, as opposed to starting execution of vertices along a new path.

#### E. Wash Queue Optimization

To facilitate cross-contamination removal, Maftei et al. [12] partition the DMFB into regions, with one wash droplet allocated to each region. Within a region, each droplet movement that leaves residue behind is recorded and appended to a list of cells (a queue) that require washing.

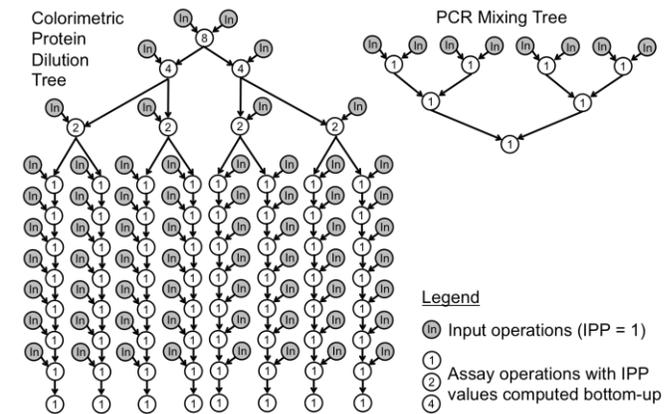


Fig. 15. A 3-level colorimetric protein dilution tree (left) and a PCR mixing tree (upper right). Vertices are labeled with their IPP values.

The wash droplet is then routed to each contaminated cell in the order in which they appear in the list. This approach works well if at most one droplet at a time contaminates a region: the list stores the cells along the path taken by the droplet. However, if multiple contaminating droplets travel through the region at the same time, the wash droplet route computed by this algorithm may be inefficient. For example, Fig. 16(a) shows two droplets moving along the left and right perimeter. Based on the scheme outlined by Maftai et al., the wash droplet repeatedly crisscrosses the region, washing one contaminated cell, per side. Fig. 16(b) depicts a more efficient wash droplet route that eliminates unnecessary crisscrossing.

Let  $P$  be a partition and  $w_p$  be its corresponding wash droplet. If no cells in  $P$  are contaminated, the corresponding wash droplet is routed to the center of the partition, where it rests until a protocol droplet enters the partition.  $P$  maintains a list  $L_{wash}(d)$  of cells within the partition that have been contaminated by protocol droplet  $d$ ; we assume that a cell becomes contaminated when  $d$  leaves it by moving to a neighbor. To avoid redundancy, cell  $c$  is only added to  $L_{wash}(d)$  when it becomes contaminated; if  $d$  returns to  $c$  after several time-steps (e.g., due to a reversal of direction), but before  $c$  has been decontaminated by  $w_p$ , then  $c$  is not added to  $L_{wash}(d)$  a second time, because it only needs to be cleaned once.

Let  $d$  be the only protocol droplet in partition  $P$ . Once the first contaminated cell  $c$  is added to  $L_{wash}(d)$ , wash droplet  $w_p$  selects  $c$  as its destination and is routed there probabilistically. Upon arriving,  $w_p$  decontaminates  $c$ ;  $w_p$  then follows  $d$  based on the ordering of cells in  $L_{wash}(d)$ :  $c$ , the first entry in  $L_{wash}(d)$  is removed, and  $w_p$  selects the following cell in  $L_{wash}(d)$  (an adjacent neighbor of  $c$ ) as its next target. To reduce the likelihood of deadlock  $w_p$  follows the cells in  $L_{wash}(d)$  probabilistically. A movement toward the first cell in  $L_{wash}(d)$ , as per the Manhattan distance from  $d$ 's location, is a positive move; neutral and negative moves are handled similarly.

If  $d$  eventually leaves  $P$ ,  $w_p$  will decontaminate all cells in  $L_{wash}(d)$ . If  $d$  stalls within  $P$  (e.g., due to congestion), then  $w_p$  may stall as well, as it cannot enter a congested cell adjacent to  $d$ , as per droplet spacing rules [33]. To reduce congestion, it may be necessary to move  $w_p$  away from  $d$ , which provides  $d$  with the opportunity to backtrack. Hence,  $w_p$  uses probabilistic routing to clean the cells in  $L_{wash}(d)$ ; if  $w_p$  followed the path of cells in  $L_{wash}(d)$ , exclusively, then it would not be possible to back off, and deadlocks would be far more likely to occur.

If droplets,  $d$  and  $d'$  enter  $P$ , then the algorithm must choose whether to first decontaminate cells in  $L_{wash}(d)$  or  $L_{wash}(d')$ , and whether or not to decontaminate *all* cells in one list before moving on to the other. If  $w_p$  follows  $d$ , and then pauses due to congestion, it can be beneficial to switch to cleaning the cells in  $L_{wash}(d')$  while removing  $d$  from the congestion region.

Fig. 17 provides an illustrative example. In Fig. 17(a), there is no congestion, so the wash droplet can clean all cells contaminated by the orange droplet before cleaning the cells contaminated by the green droplet. In Fig. 17(b), congestion stops the orange droplet at the partition exit. Rather than waiting for the orange droplet to move, the wash droplet cleans the cells contaminated by the green droplet, as shown in Fig. 17(c); this opens up a path by which the orange droplet can move out of the congested region; the wash droplet then cleans the remaining cells contaminated by the orange droplet.

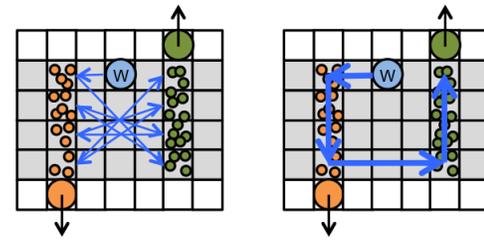


Fig. 16. (a) When multiple droplets travel concurrently through a region allocated to a wash droplet, the washing scheme proposed by Maftai et al. [12] may yield highly inefficient routes due to the order in which cells to wash are inserted into a list. (b) a shorter and more concise droplet route as obtained by our enhanced implementation of washing for routing-based synthesis.

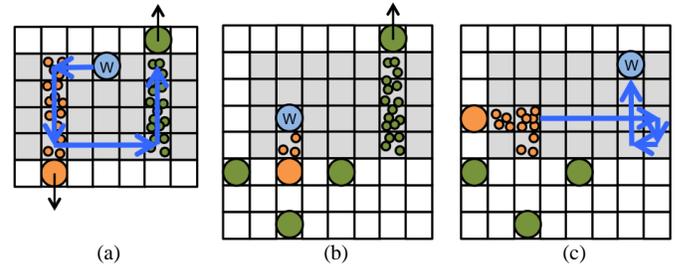


Fig. 17. Wash droplet routing when multiple droplets contaminate a partition. (a) The ideal case, in which the wash droplet can wash one path, followed by the other; (b) a more complex case, where the wash droplet cannot fully wash one path because the droplet that causes the contamination is blocked due to congestion; (c) after partially decontaminating one path, the wash droplet cleans the other, allowing the stuck droplet to escape from the congested area.

We consider  $w_p$  to be congested if it follows a path of cells in  $L_{wash}(d)$ , but does not advance for  $T$  time-steps, where  $T$  is a threshold. If  $w_p$  is congested, then it becomes free to select another list of contaminated cells  $L_{wash}(d')$  to follow. If there are multiple available lists, the one whose first cell is closest to  $w_p$ 's current position, in terms of Manhattan distance, is chosen. This approach readily generalizes to any number of protocol droplets in a partition.

## V. SIMULATION RESULTS

We implemented our routing-based synthesis algorithms in a publicly available open source DMFB synthesis tool [42]. All simulations were run on a 15x9 DMFB with six 3x4 detection modules. We assume a 100 Hz actuation frequency for droplet movements; in other words, it takes 10ms to move a droplet from its current position to a neighboring cell. When washing is enabled, we use 9 wash droplets with capacity of 1024, unless stated otherwise; when washing is disabled, the simulator ignores contamination and droplets may cross paths at-will. Starting with the baseline method we offer five enhancements that can be independently enabled or disabled, yielding  $2^5 = 32$  algorithmic configurations when washing is enabled, and  $2^4 = 16$  configurations when washing is disabled.

**Base:** The baseline routing-based synthesis algorithm [12] (Fig. 10) extended to utilize the six detectors (Section IV.B).

**In:** Input reservoir congestion alleviation (Figs 11 and 12); a 2x3 region surrounding each input reservoir is reserved to ensure that droplets can safely enter the board; this reduces the amount of spatial parallelism on the chip.

**Out:** Output reservoir congestion alleviation (Fig. 13); a 2x3 region surrounding each output reservoir is reserved to reduce the likelihood of congestion-induced deadlock; this also reduces the amount of spatial parallelism on the chip.

**OdP:** Dynamic adjustment of droplet movement probabilities based on Operation-driven Profitability (Section IV.C).

**IPP:** Independent path priority (IPP)-based congestion-aware scheduling (Section IV.D).

**WQ:** Wash Queue optimization (Section IV.E).

For example, **In-Out-OdP** represents a run with the **In**, **Out**, and **OdP** enhancements enabled, and **IPP** and **WQ** disabled.

For each experiment (assay/algorithmic configuration), we perform multiple runs (1000, 100, or 50, depending on the size of the assay) with different random number seeds. First, we report the success rate (the percentage of runs that execute to completion). For the successful runs, we report the average assay execution time (time-steps), the standard deviation, and the minimum and maximum execution times.

The overall quality of results depends on the context in which the algorithm is employed. In an offline context, there may be ample time to perform hundreds or thousands of runs using multiple algorithmic configurations, in search of one solution that minimizes the overall execution time. In an online context, where computation time is limited and the user expects real-time execution of an assay, the most important metrics are the success rate, average execution time, and standard deviation. Presumably, large variations in assay execution time for successful runs are due to the probabilistic occurrence of localized congestion, and the average overhead due to detection and recovery (e.g., **OdP**). Without **OdP**, deadlocks are much more likely, as reported in our results.

In principle, online deadlock recovery could be achieved by merging and contaminating some droplets, moving the contaminated droplets to a waste reservoir, and invoking an error recovery procedure [7]; however, this would increase execution time significantly. We do not investigate this here.

#### A. PCR Mixing Tree

The first experiment evaluates routing-based synthesis on a 7-node, 3-level PCR mixing tree assay (PCR) [18]; see Fig. 18 (right). We performed 1000 runs for each configuration with washing disabled, none of which deadlocked, due to the small size of the assay relative to the spatial area of the DMFB.

Table II presents the results of this experiment. The PCR mixing tree is relatively small, so there is minimal competition for spatial DMFB resources. As a result: **IPP** offers no improvement over the baseline; **OdP** alleviates some localized congestion, thus improving all metrics; **IPP-OdP** and **In-Out-IPP-OdP** offers comparable results to **OdP**.

Table III presents similar results when washing is enabled: **IPP** is ineffective; **OdP-WQ** reduces the average execution time and best/worst-case results, but increases the standard deviation, as the best-case results are significantly better than for **IPP**; **IPP-OdP-WQ** and **In-Out-IPP-OdP-WQ** yield comparable results to **OdP-WQ**. Even with 9 droplets on the board, there were no significant resource constraints.

TABLE II  
ASSAY EXECUTION TIMES (TIME-STEPS) FOR DIFFERENT ALGORITHMIC CONFIGURATIONS OF ROUTING-BASED SYNTHESIS ON THE PCR ASSAY (WASHING DISABLED); ALL RUNS COMPLETED SUCCESSFULLY.

	Base	IPP	OdP	IPP-OdP	In-Out-IPP-OdP
Avg. Exec. Time	874.62	875.10	514.57	514.75	514.83
Std. Dev.	27.54	28.69	13.49	13.58	13.58
Min.	795	794	476	473	473
Max.	973	1010	559	560	560

TABLE III  
ASSAY EXECUTION TIMES (TIME-STEPS) FOR DIFFERENT ALGORITHMIC CONFIGURATIONS OF ROUTING-BASED SYNTHESIS ON THE PCR ASSAY (WASHING ENABLED)

	Base	IPP	OdP-WQ	IPP-OdP-WQ	In-Out-IPP-OdP-WQ
Avg. Exec Time	1223.06	1225.00	755.37	755.45	754.45
Std. Dev.	69.40	69.17	82.50	84.73	84.80
Min.	1047	1056	623	618	607
Max.	1944	1598	1310	1310	1218

To summarize, the results reported in Tables II and III show indicate that only **OdP** is effective in the absence of resource constraints, as it detects and alleviates localized congestion.

#### B. 5-Level Exponential Protein Dilution Assay

The second experiment considers a colorimetric protein dilution tree with 5-levels of splitting at the top (PS-5) [14]; for reference, the tree in Fig. 15 only has 3 levels. The PS-5 DAG has 543 vertices: 256 inputs, 32 splits, 224 mix/dilution operations, and 32 detection operations. Referring back to Fig. 15, the detection operations are the leaf nodes of the DAG, i.e., those without successors. We performed 100 runs, with and without washing, for each algorithmic configuration. Tables IV and V report the results of this experiment.

With washing enabled (Table V), **Base** achieved a miserable 10% success rate; adding **IPP** and **OdP** in isolation increased the success rate to 25% and 80% respectively. Adding **Input** and **Output** reservoir congestion alleviation increased the success rates to 100%, and, once again, the combination of all four enhancements (other than **WQ**) yielded the best results in terms of assay execution time; a detailed analysis of **WQ** is deferred to the next subsection.

TABLE IV  
SUCCESS RATE AND ASSAY EXECUTION TIMES (TIME-STEPS) FOR DIFFERENT ALGORITHMIC CONFIGURATIONS OF ROUTING-BASED SYNTHESIS ON THE PS-5 ASSAY (WASHING DISABLED)

	Base	IPP	OdP	In-Out-OdP	In-Out-IPP-OdP
Success Rate	100%	100%	100%	100%	100%
Avg. Exec. Time	26276.02	22540.70	25155.05	24382.17	21889.69
Std. Dev.	14854.91	1399.74	5318.47	3953.71	1251.98
Min.	22313	21841	20744	20825	20272
Max.	140681	34590	51453	51785	26269

TABLE V  
SUCCESS RATE AND ASSAY EXECUTION TIMES (TIME-STEPS) FOR DIFFERENT ALGORITHMIC CONFIGURATIONS OF ROUTING-BASED SYNTHESIS ON THE PS-5 ASSAY (WASHING ENABLED)

	Base	IPP	OdP	In-Out-OdP	In-Out-IPP-OdP
Success Rate	10%	25%	80%	100%	100%
Avg. Exec. Time	85344.60	70254.40	49737.21	54704.39	52072.35
Std. Dev.	33332.01	34616.98	7263.57	5024.22	4450.42
Min.	52992	34370	35472	43990	42468
Max.	165831	167109	73542	66998	67274

The introduction of wash droplets (9 in this experiment) increased competition for limited spatial resources; a separate experiment using the **In** configuration (not shown in Table V) achieved a 100% success rate, indicating that the situations shown in Figs. 11 and 12 cause a significant number of deadlocks; the situation shown in Fig. 13 did not occur in these experiments (an unreported experiment using the **Out** configuration had an 8% success rate); however, we did observe this situation at many points during the process of developing and refining the enhancements reported here.

Unlike Table IV, **OdP**, rather than **IPP**, was responsible for the most significant reductions in the standard deviation and maximum assay execution times reported over the 100 runs. This result suggests that **OdP** has a much greater impact when there is greater competition for spatial resources (due to the 9 wash droplets in conjunction with the sufficiently large PS-5 assay). That being said, **IPP** still offers some performance benefits: for example, **In-Out-OdP-IPP** reduced the average assay execution time by 4.8% compared to **In-Out-OdP**.

**OdP** (considering only the successful runs) attained the lowest average execution time, while **IPP** obtained the fastest individual run. These two algorithms exhibited 80% and 25% success rates, respectively, along with very high standard deviations, rendering them useful in an offline context. These configurations may be capable of finding very good quality solutions for probabilistically “lucky” runs where I/O reservoir congestion does not occur. For these runs, the **In** and **Out** configuration options offer no benefit while consuming spatial resources that could otherwise be used for operations.

### C. Impact of Wash Queue Optimization

Our next experiment evaluates the impact of the **WQ** configuration on the success rate and assay execution time of PS-5 on a 9x15 DMFB, with 3, 6, and 9 wash droplets using two algorithmic configurations: **In-Out-IPP-OdP** and **In-Out-IPP-OdP-WQ**. 100 runs were performed for each configuration (algorithmic + number of wash droplets). Table VI reports the results of this experiment.

Success rates of 100% were achieved with 6 and 9 wash droplets, regardless of whether or not **WQ** is enabled. For 6 wash droplets, enabling **WQ** reduced the average assay execution time by 3.1%; for 9 wash droplets, enabling **WQ** increased the average assay execution time by a statistically insignificant factor of 0.76%. As the number of wash droplets on-chip increases, the likelihood of multiple assay droplets simultaneously traveling through the region assigned to a given wash droplet decreases; thus, the impact of **WQ** lessens; at the same time, increasing the number of wash droplets on-chip benefits all assay execution time metrics, and lowers the standard deviation among execution times.

With 3 wash droplets, **WQ** improves all reported metrics. These results suggest that inefficient washing (without **WQ**) leads to deadlocks that are induced, in part, by reduced spatial parallelism, as contaminated cells are unusable; additionally, wash droplets may themselves be deadlocked, and are thus unable to clean contaminated regions of the chip.

These results indicate that increasing the number of wash droplets on-chip increases overall performance. Clearly, there must be a limit to this benefit, as an exorbitant number of wash droplets would cause congestion and lead to deadlocks.

TABLE VI

SUCCESS RATE AND ASSAY EXECUTION TIMES (TIME-STEPS) FOR DIFFERENT ALGORITHMIC CONFIGURATIONS OF ROUTING-BASED SYNTHESIS ON THE PS-5 ASSAY WHILE VARYING THE NUMBER OF WASH DROPLETS. (IN-OUT-IPP-OdP vs. IN-OUT-IPP-OdP-WQ)

	3 Wash Droplets		6 Wash Droplets		9 Wash Droplets	
	No WQ	WQ	No WQ	WQ	No WQ	WQ
Success Rate	71%	97%	100%	100%	100%	100%
Avg. Exec. Time	138944.06	90173.36	66321.85	64256.21	52072.35	52466.55
Std. Dev.	28674.83	11682.43	8171.00	7338.67	4450.42	4001.61
Min.	85603	66966	49038	49315	42468	44759
Max.	193446	149033	93667	83266	67274	62286

TABLE VII

SUCCESS RATES AND ASSAY EXECUTION TIMES (TIME-STEPS) FOR A 7-LEVEL EXPONENTIAL PROTEIN DILUTION ASSAY (IPP ENABLED/DISABLED)

	In-Out-OdP-WQ	In-Out-IPP-OdP-WQ
Success Rate	96%	96%
Avg. Exec. Time	259734.24	211201.39
Std. Dev.	16923.82	7906.50
Min.	228583	198629
Max.	291050	231143

In practice, this limit would depend on the DMFB area and the assay’s demand for spatial resources, however, it is not our objective to quantify this limit here. Meanwhile, decreasing the number of wash droplets increases the favorable impact of **WQ**. As wash droplets are relatively cheap compared to samples and reagents [43, 44], we suspect that a typical user would opt for a relatively large number of wash droplets on-chip (i.e., 9 in this case) in order to optimize performance.

### D. 7-Level Exponential Protein Dilution Assay

This experiment considers a 7-level exponential protein dilution assay (PS-7) on a 9x15 DMFB with washing enabled to compare the **In-Out-OdP-WQ** and **In-Out-IPP-OdP-WQ** configurations. We ran each configuration 50 times due to the large assay. Table VII reports the result of this experiment.

The success rate of **IPP** was 96% in both cases, suggesting that it should be viewed as a performance optimization, not a deadlock prevention strategy. **IPP** improved all execution time metrics: across the 50 runs, **IPP** reduced average execution time by 19%, the standard deviation by 53%, and the minimum and maximum execution times by 13% and 21%, respectively. By limiting the number of concurrent operations based on the availability of on-chip resources, **IPP** enables the operations that *are* executing to pick more profitable moves; without **IPP**, more operations execute concurrently, but they choose the most profitable moves with much lower frequency, thus increasing assay execution time. These results confirm the observations about **IPP** from Subsection V.B.

### E. Impact of Wash Droplet Capacity

In this experiment, we ran the PS-5 assay on a 9x15 DMFB using the **In-Out-IPP-OdP-WQ** configuration with 9 wash droplets using the following wash droplet capacities: {16, 32, 64, 128, 256, 512, 1024, 2048, 4096}. 100 runs were performed for each wash droplet capacity. Fig. 18 reports the minimum, average, and maximum, assay execution time for each capacity. Four of the 900 runs in this experiment failed: one each for capacities of 16 and 512, and two for capacities of 64. Failed runs do not contribute to the averages in Fig. 18.

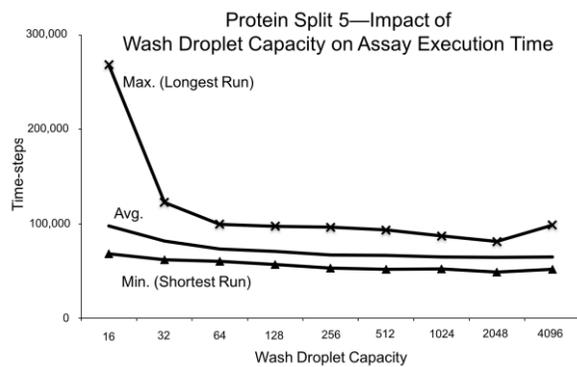


Fig. 18. Impact of wash droplet capacity on execution time (time-steps) on the PS-5 assay. 100 runs were using different random number seeds at each wash droplet capacity; for each capacity, the maximum, minimum, and average assay execution time are reported.

Fig. 18 shows that the wash droplet capacity affects the maximum assay execution time, especially at lower capacities (16, 32) far more than it affects the average or minimum assay execution times at all capacities. For wash droplet capacities that exceed 64, the relative impact on minimum or average assay execution time is quite small.

In a realistic deployment scenario, the washing capacity of a droplet will depend on physical properties of the DMFB, samples and reagents used in the experiment, and the chosen washing solution. If the cost of the washing solution is an issue, then the benefits of choosing a solution that can clean more than 64 cells per droplet is marginal at best.

#### F. Comparison with Module-based Synthesis

Lastly, we perform a scalability study that compares the performance of routing-based synthesis with the traditional approach of module-based synthesis. The benchmarks used in the study are a sequence of exponential protein dilution assays,  $\{PS-k \mid 1 \leq k \leq 7\}$  (“PS” stands for “Protein Split”).  $PS-k$  has  $k$  levels of splitting (e.g., the colorimetric protein dilution tree in Fig. 15 is PS-3), and produces a tree with  $2^k$  paths emanating from the root node; thus, this particular class of assays exhibits exponential growth in terms of parameter  $k$ .

For module-based synthesis, we use Path Scheduling [13], placement based on a Virtual Topology [30], and a maze routing algorithm introduced by Roy et al. [38]. We selected Path Scheduling because it outperformed other heuristics for the  $PS-k$  family of benchmarks in prior studies [14, 15].

We ran routing-based synthesis with and without wash droplet routing. With washing disabled, we used the **In-Out-OdP-IPP** configuration, and with washing enabled, we used the **In-Out-OdP-IPP-WQ** configuration. We performed 100 runs for PS-1, PS-2, and PS-3, 75 runs for PS-4, PS-5 and PS-6, and 50 runs for PS-7; we report the minimum (Min.), maximum (Max.) and average (Avg.) execution times for each benchmark, with and without washing enabled. Fig. 19 compares these results with module-based synthesis.

Module-based synthesis exhibited longer execution times than routing-based synthesis without washing for all seven of the  $PS-k$  benchmarks. The performance gap starts narrow, and becomes more pronounced as  $k$  increases, with a dramatic widening at PS-7.

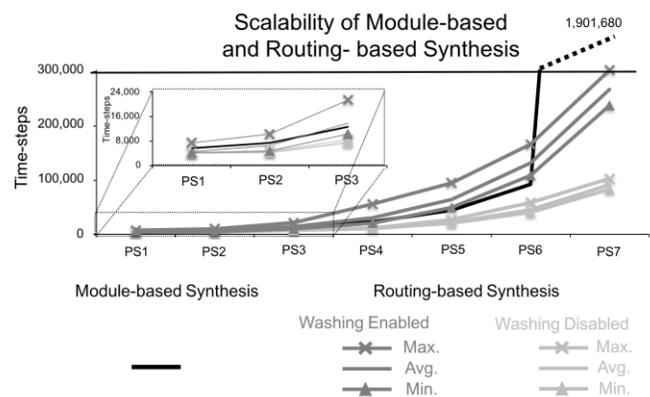


Fig. 19. Comparison between module-based and routing-based synthesis on a family of exponential protein dilution trees.

Module-based synthesis approximately tracks the average execution time of routing-based synthesis with washing enabled for  $PS-1..3$ . At  $PS-3$ , the execution time curves cross; module-based synthesis outperforms routing-based synthesis with washing from  $PS-3..6$ , with a clear improvement over the best routing-based synthesis run (Min.) at  $PS-6$ . Cross-contamination and the presence of wash droplets on-chip increase operation latencies for routing-based synthesis in three ways: (1) there are more droplets on-chip due to the 9 wash droplets; (2) operation latencies increase due to pauses waiting for contamination removal; and (3) contaminated cells reduce the available area for droplet movement. Module-based synthesis does not suffer these drawbacks, as contamination is removed during routing [39-41], and assay execution time is dominated by the schedule, not routing [13, 33].

For  $PS-7$ , the execution latency of module-based synthesis is 7.1x longer than the average execution time of routing-based synthesis with washing enabled. Recall that we use a  $15 \times 9$  DMFB for all benchmarks. Routing-based synthesis retains two advantages: (i) the module abstraction limits the spatial parallelism available to the scheduler for module-based synthesis; in contrast, the congestion-aware scheduling mechanism employed by routing-based synthesis is more aggressive and enables routing-based synthesis to execute more operations in parallel; and (ii) routing-based synthesis has the probabilistic opportunity to execute faster operations by choosing more profitable movements than the limited set of movements allowable within a module. Thus, when on-chip resource constraints become stringent, routing-based synthesis outperforms module-based synthesis.

## VI. CONCLUSION

Routing-based synthesis, as initially conceived [12], did not support non-reconfigurable operations and was susceptible to both livelock and deadlock (Figs. 11-13), often occurring in locally congestion regions of a DMFB. This paper overcomes these drawbacks through: (i) the introduction of algorithms to support non-reconfigurable operations; (ii) techniques to detect operations slowed by congestion; and (iii) techniques to alleviate localized livelock and deadlock by dynamically throttling the droplet movement probabilities. Routing-based synthesis yields faster assay execution times than module-based synthesis when parallelism is limited.

## REFERENCES

- [1] M. J. Jebraill, M.S. Bartsch, and K. D. Patel. (2012, Jul.). Digital microfluidics: a versatile tool for applications in chemistry, biology, and medicine. *Lab-on-a-Chip* [Online]. 12(14), pp. 5452-2463. Available: <http://dx.doi.org/10.1039/C2LC40318H>
- [2] M. G. Pollack, A. D., Shenderov, and R. B. Fair. (2002, Mar.). Electrowetting-based actuation of droplets for integrated microfluidics. *Lab-on-a-Chip* [Online]. 2(2), pp. 96-101. Available: <http://dx.doi.org/10.1039/b110474h>
- [3] J. H. Noh, J. Noh, E. Kreit, J. Heikenfeld, and P. D. Rack (2012, Feb.) Toward active-matrix lab-on-a-chip: programmable electrofluidic control enabled by arrayed oxide thin film transistors. *Lab-on-a-Chip* [Online]. 12(2), pp. 353-360. Available: <http://dx.doi.org/10.1039/c1lc20851a>
- [4] B. Hadwen, G. R. Broder, D. Morganti, A. Jacobs, C. Brown, J. R. Hector, Y. Kubota, and H. Morgan (May, 2012). Programmable large-area digital microfluidic array with integrated droplet sensing for bioassays. *Lab-on-a-Chip* [Online]. 12(18), pp. 3305-3313. Available: <http://dx.doi.org/10.1039/c2lc40273d>
- [5] Y. Luo, B. B. Bhattacharya, T-Y. Ho, and K. Chakrabarty, "Optimization of polymerase chain reaction on a cyberphysical digital microfluidic biochip," In *Proc. ICCAD*, San Jose, CA, USA, 2013, pp. 622-629. Available: <http://dx.doi.org/10.1109/ICCAD.2013.6691181>
- [6] Y. Luo, K. Chakrabarty, and T-Y. Ho. (2013, Jan.) Error recovery in cyberphysical digital microfluidic biochips. *IEEE Trans CAD* [Online]. 32(1), pp. 59-72. Available: <http://dx.doi.org/10.1109/TCAD.2012.2211104>
- [7] Y. Zhao, T. Xu, and K. Chakrabarty. (2010, Aug.) Integrated control-path design and error recovery in the synthesis of digital microfluidic biochips. *ACM Journal on Emerging Technologies in Computing Systems* [Online]. 6(3), article #11. Available: <http://dx.doi.org/10.1145/1777401.1777404>
- [8] M. A. Murran, and H. Najjaran. (2012, Mar.). Capacitance-based droplet position estimator for digital microfluidic devices. *Lab-on-a-Chip* [Online]. 12(11), pp. 2053-2059. Available: <http://dx.doi.org/10.1039/c2lc21241b>
- [9] S. C. C. Shih, et al. (2013, Apr.). Digital microfluidics with impedance sensing for integrated cell culture and analysis. *Biosensors and Bioelectronics* [Online]. 42(4), pp. 314-320. Available: <http://dx.doi.org/10.1016/j.bios.2012.10.035>
- [10] K. Choi, et al. (2013, Aug.). Automated digital microfluidic platform for magnetic-particle-based immunoassays with optimization by design of experiments. *Analytical Chemistry* [Online]. 85(20), pp. 9638-9646. Available: <http://dx.doi.org/10.1021/ac401847x>
- [11] P. Paik, V. Pamula, and R. Fair. (2003, Sep.) Rapid droplet mixers for digital microfluidic systems. *Lab-on-a-Chip* [Online]. 3(4), pp. 253-259. Available: <http://dx.doi.org/10.1039/b307628h>
- [12] E. Maftai, P. Pop, and J. Madsen. (2012, Mar.). Routing-based synthesis of digital microfluidic biochips. *Springer Journal of Design Automation for Embedded Systems* [Online]. 16(1), pp. 19-44. Available: <http://dx.doi.org/10.1007/s10617-012-9083-0>
- [13] F. Su, and K. Chakrabarty. (2008, Jan). High-level synthesis of digital microfluidic biochips. *ACM Journal on Emerging Technologies in Computing Systems* [Online]. 3(4): article #16. Available: <http://dx.doi.org/10.1145/1324177.1324178>
- [14] D. Grissom, and P. Brisk, "Path scheduling on digital microfluidic biochips," in *Proc. DAC*, San Francisco, CA, USA, 2012, pp. 26-35. Available: <http://dx.doi.org/10.1145/2228360.2228367>
- [15] K. O'Neal, D. Grissom, and P. Brisk, "Force-directed list scheduling for digital microfluidic biochips," in *Proc. VLSI-SoC*, Santa Cruz, CA, USA, 2012, pp. 7-11. Available: <http://dx.doi.org/10.1109/VLSI-SoC.2012.6378997>
- [16] C-H. Liu, K-C. Liu, and J-D. Huang, "Latency-optimization synthesis with module selection for digital microfluidic biochips," in *Proc. SOCC*, Erlangen, Germany, 2013, pp. 159-164. Available: <http://dx.doi.org/10.1109/SOCC.2013.6749681>
- [17] A. J. Ricketts, K. Irick, N. Vijaykrishnan, and M. J. Irwin, "Priority scheduling in digital microfluidics-based biochips," in *Proc. DATE*, Munich, Germany, 2006, pp. 1-6. Available: <http://dx.doi.org/10.1109/DATE.2006.244178>
- [18] J. Ding, K. Chakrabarty, and R. B. Fair. (2001, Dec.). Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *IEEE Trans CAD* [Online]. 20(12), pp. 1463-1468. Available: <http://dx.doi.org/10.1109/43.969439>
- [19] T. Xu and K. Chakrabarty. (2008, Aug.). Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips. *ACM Journal on Emerging Technologies in Computing Systems* [Online]. 4(3), article #11. Available: <http://dx.doi.org/10.1145/1389089.1389091>
- [20] T. Xu, K. Chakrabarty, and F. Su. (2008, Mar.) Defect-aware high-level synthesis and module placement for microfluidic biochips. *IEEE Trans. Biomedical Circuits and Systems* [Online]. 2(1), pp. 50-62. Available: <http://dx.doi.org/10.1109/TBCAS.2008.918283>
- [21] E. Maftai, P. Pop, and J. Madsen (2010, Jul.). Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices. *Springer Journal of Design Automation for Embedded Systems* [Online]. 14, pp. 287-308. Available: <http://dx.doi.org/10.1007/s10617-010-9059-x>
- [22] E. Maftai, P. Pop, and J. Madsen (2013, Feb.) Module-based synthesis of digital microfluidic biochips with droplet-aware operation execution. *ACM Journal on Emerging Technologies in Computing Systems* [Online]. 4(3), article #11. Available: <http://dx.doi.org/10.1145/2422094.2422096>
- [23] K. Bazargan, R. Kastner, and M. Sarrafzadeh. (2000, Jan.-Mar.). Fast template placement for reconfigurable computing. *IEEE Design and Test of Computers* [Online]. 17(1), pp. 68-83. Available: <http://dx.doi.org/10.1109/54.825678>
- [24] C. C-Y. Lin, and Y-W. Chang. (2011, Jun.). Cross-contamination aware design methodology for pin-constrained digital microfluidic biochips. *IEEE Trans CAD* [Online]. 30(6), pp. 817-828. Available: <http://dx.doi.org/10.1145/1837274.1837438>
- [25] F. Su and K. Chakrabarty. (2006, Jul.). Module placement for fault-tolerant microfluidics-based biochips. *ACM Trans. Design Autom. Electron. Syst* [Online]. 11(3), pp. 687-710. Available: <http://dx.doi.org/10.1145/1142980.1142987>
- [26] C. Liao and S. Hu. (2011, Mar.). Multiscale variation-aware techniques for high-performance digital microfluidic lab-on-a-chip component placement. *IEEE Trans. Nanobioscience* [Online]. 10(1), pp. 51-58. Available: <http://dx.doi.org/10.1109/TNB.2011.2129596>
- [27] E. J. Griffith, S. Akella, and M. K. Goldberg. (2006, Feb.) Performance characterization of a reconfigurable planar-array digital microfluidic system. *IEEE Trans CAD* [Online]. 25(2), pp. 345-357. Available: <http://dx.doi.org/10.1109/TCAD.2005.859515>
- [28] D. Grissom and P. Brisk, "A high-performance online assay interpreter for digital microfluidic biochips," in *Proc. GLSVLSI*, Salt Lake City, Utah, USA, 2012, pp. 103-106. Available: <http://dx.doi.org/10.1145/2206781.2206808>
- [29] D. Grissom and P. Brisk, "A field-programmable pin-constrained digital microfluidic biochip," in *Proc. DAC*, Austin, TX, USA, 2013, article #46. Available: <http://dx.doi.org/10.1145/2463209.2488790>
- [30] D. Grissom and P. Brisk. (2014, Mar.). Fast online synthesis of digital microfluidic biochips. *IEEE Trans CAD* [Online]. 33(3), pp. 356-369. Available: <http://dx.doi.org/10.1109/TCAD.2013.2290582>
- [31] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. (2007, Nov.) Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation. *ACM Journal on Emerging Technologies in Computing Systems* [Online]. 3(3), pp. 13.1-13.32. Available: <http://dx.doi.org/10.1145/1295231.1295234>
- [32] Y-H. Chen, C-L. Hsu, L-C. Tsai, T-W. Huang, and T-Y. Ho. (2013, Aug.). A reliability-oriented placement algorithm for reconfigurable digital microfluidic biochips using 3-D deferred decision-making technique. *IEEE Trans CAD* [Online]. 32(8), pp. 1151-1162. Available: <http://dx.doi.org/10.1109/TCAD.2013.2249558>
- [33] F. Su, W. Hwang, and K. Chakrabarty, "Droplet routing in the synthesis of digital microfluidic biochips," in *Proc. DATE*, Munich Germany,

- 2006, pp. 323-328. Available: <http://dx.doi.org/10.1109/DATE.2006.244177>
- [34] K. F. Böhringer. (2006, Feb.). Modeling and controlling parallel tasks in droplet-based microfluidic systems. *IEEE Trans CAD* [Online]. 25(2), pp. 334-344. Available: <http://dx.doi.org/10.1109/TCAD.2005.855958>
- [35] M. Cho and D. Z. Pan. (2008, Oct.). A high-performance droplet routing algorithm for digital microfluidic biochips. *IEEE Trans CAD* [Online]. 27(10), pp. 1714-1724. Available: <http://dx.doi.org/10.1145/1353629.1353672>
- [36] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. (2008, Nov.) BioRoute: a network flow- based routing algorithm for the synthesis of digital microfluidic biochips. *IEEE Trans CAD* [Online]. 27(11), pp. 1928–1941. Available: <http://dx.doi.org/10.1109/TCAD.2008.2006140>
- [37] T. Huang and T. Ho, “A fast routability- and performance-driven droplet routing algorithm for digital microfluidic biochips,” in *Proc. ICCD*, Lake Tahoe, CA, USA, 2009, pp. 445–450. Available: <http://dx.doi.org/10.1109/ICCD.2009.5413119>
- [38] P. Roy, H. Rahaman, and P. Dasgupta, “A novel droplet routing algorithm for digital microfluidic biochips,” in *Proc. GLSVLSI*, Providence, RI, USA, 2010, pp. 441-446. Available: <http://dx.doi.org/10.1145/1785481.1785583>
- [39] T-W. Huang, C-H. Lin, and T-Y. Ho. (2010, Nov.). A contamination aware droplet routing algorithm for the synthesis of digital microfluidic biochips. *IEEE Trans CAD* [Online]. 29(11), pp. 1682–1695. Available: <http://dx.doi.org/10.1109/TCAD.2010.2062770>
- [40] Y. Zhao and K. Chakrabarty. (2012, Jun.). Cross-contamination avoidance for droplet routing in digital microfluidic biochips. *IEEE Trans CAD* [Online]. 31(6), pp. 817–830. Available: <http://dx.doi.org/10.1109/TCAD.2012.2183369>
- [41] Q. Wang, Y. Shen, H. Yao, T.-Y. Ho, and Y. Cai. (2014, Jun.) Practical functional and washing droplet routing for cross-contamination avoidance in digital microfluidic biochips. in *Proc. DAC*, San Francisco, CA, USA, 2014, pp. 1-6. Available: <http://dx.doi.org/10.1145/2593069.2593189>
- [42] D. Grissom, K. O’Neal, B. Preciado, H. Patel, R. Dohert, N. Liao, and P. Brisk (2012, Oct.) “A digital microfluidic biochip synthesis framework, in *Proc. VLSI-SoC*, Santa Cruz, CA, USA, 2012, pp. 177-182. Available: <http://dx.doi.org/10.1109/VLSI-SoC.2012.6379026>
- [43] Y-L. Hsieh, T-Y. Ho, and K. Chakrabarty. (2012, Nov.). A reagent-saving mixing algorithm for preparing multi-target biochemical samples using digital microfluidics. *IEEE Trans CAD* [Online]. 31(11), pp. 1556-1669. Available: <http://dx.doi.org/10.1109/TCAD.2012.2202396>
- [44] J-D. Huang, C-H. Liu, and T-W. Chiang, “Reactant minimization during sample preparation on digital microfluidic biochips using skewed mixing trees,” In *Proc. ICCAD*, San Jose, USA, 2012, pp. 377-383. Available: <http://dx.doi.org/10.1145/2429384.2429464>

**Skyler Windh** received the B.S. Degree in Computer Science from the University of California, Riverside (UCR). He is presently pursuing a Ph.D. degree in Computer Science at UCR. His research interests include the usage of FPGAs and GPUs to accelerate database and data mining applications.

**Calvin Phung** received the B.S. Degree in Computer Science from the University of California, Riverside (UCR) in 2012. He is presently pursuing a Ph.D. degree in Computer Science at UCR. He is the leading engineer for the first brain-training game for the Brain Game Center at UCR, a collaborative project between the Computer Graphics and the Cognitive Neuroscience Labs. His research interests include machine learning to predict the performance in future sessions for general cognitive training applications to provide challenging, but not overwhelming, application experiences for the user.

**Daniel Grissom** received the B.S. degree in computer engineering from the University of Cincinnati in 2008, and the M.S. and Ph.D. degrees in computer science from the University of California, Riverside, in 2011 and 2014 respectively. He is now an Assistant Professor at Azusa Pacific University.

**Paul Pop (M’99)** is a professor at DTU Compute, Technical University of Denmark (DTU). He has received his Ph.D. degree in computer systems from Linköping University in 2003. His main research interests are in the area of system-level design of embedded systems. He has published extensively in this area, and has received the best paper award at the DATE 2005, RTiS 2007, CASES 2009 and MECO 2013 conferences and the EDAA Outstanding Dissertations Award (co-supervisor) in 2011. Since 2008 he has also addressed Computer-Aided Design methods for biochips. His work in this area has received the best paper award at the CASES 2009 conference. He has co-organized and participated in tutorials and special sessions on CAD for biochips at conferences such as SOCC 2011, ESWEEK 2011, EMBC 2015 and ETS 2015.

**Philip Brisk (M’09)** received the B.S., M.S., and the Ph.D. degrees, all in computer science, from the University of California, Los Angeles, Los Angeles, CA, USA, in 2002, 2003, and 2006, respectively.

From 2006 to 2009, he was a Post-Doctoral Scholar at the Processor Architecture Laboratory, School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. He is an Associate Professor at the Department of Computer Science and Engineering, the University of California, Riverside, Riverside, CA, USA. His current research interests include programmable microfluidics, FPGAs, compilers, and design automation and architecture for application-specific processors.

Dr. Brisk was the recipient of the Best Paper Award at CASES, 2007 and FPL 2009. He has been a Program Committee Member for several international conferences and workshops, including DAC, ASPDAC, DATE, VLSI-SoC, FPL, FPT, and so on. He has been a General (co-)Chair of IEEE SIES 2009, IEEE SASP 2010, and IWLS 2011, and Program (co-)Chair of IEEE SASP 2011, IWLS 2012, ARC 2013, and FPL 2016.