A Performance-Optimizing Compiler for Cyber-Physical Digital Microfluidic Biochips

Tyson Loveless
University of California, Riverside
USA
tlove004@ucr.edu

Jason Ott
University of California, Riverside
USA
jott002@ucr.edu

Philip Brisk
University of California, Riverside
USA
philip@cs.ucr.edu

Abstract

This paper introduces a compiler optimization strategy for Software-Programmable Laboratories-on-a-Chip (SP-LoCs), which miniaturize and automate a wide variety of benchtop laboratory experiments. The compiler targets a specific class of SP-LoCs that manipulate discrete liquid droplets on a 2D grid, with cyber-physical feedback provided by integrated sensors and/or video monitoring equipment. The optimization strategy employed here aims to reduce the overhead of transporting fluids between operations, and explores tradeoffs between the latency and resource requirements of mixing operations: allocating more space for mixing shortens mixing time, but reduces the amount of spatial parallelism available to other operations. The compiler is empirically evaluated using a cycle-accurate simulator that mimics the behavior of the target SP-LoC. Our results show that a coalescing strategy, inspired by graph coloring register allocation, effectively reduces droplet transport latencies while speeding up the compiler and reducing its memory footprint. For biochemical reactions that are dominated by mixing operations, we observe a linear correlation between a preliminary result using a default mixing operation resource allocation and the percentage decrease in execution time that is achieved via resizing.

Keywords laboratory-on-a-chip (LoC), microfluidics, digital microfluidic biochip (DMFB), interference graph, coalescing

1 Introduction

The past 20 years have witnessed the development of programmable, integrated micro-scale machines called laboratories-on-a-chip (LoCs), which can automate and miniaturize a number of laboratory functions which were previously performed by hand at the benchtop scale. While the majority of LoCs that are in use today are application-specific, single-use, and disposable, software-programmable (and reusable) LoCs (SP-LoCs) are also available. At present, SP-LoCs are programmed at a level of abstraction akin to machine code, i.e., by specifying a sequence of actuation and deactuation operations for each programmable element. Moreover, many "cyber-physical" SP-LoCs feature integrated sensors, which provide feedback to the software controlling them in real-time. Thus, language and compiler support can help SP-LoCs gain traction and grow a user base. Prior work has made progress toward compiling high level languages for execution on SP-LoCs, but has either been limited in scope to a single basic block, or altogether missed optimization opportunities that can speed up compilation and dramatically decrease execution time. This paper describes solutions to these shortcomings as an optimizing compiler targeting a class of SP-LoCs called Digital Microfluidic Biochips (DMFBs). The optimization strategy crosses basic block boundaries by modeling placement of microfluidic operations on a reconfigurable processing array as a problem that generalizes graph coalescing, similar to graph coloring register allocation in traditional compilers. Fluid transport operations can be eliminated through a coalescing mechanism; when coalescing is not possible, fluid transport lengths can be reduced by incorporating knowledge of transport operations into placement. The compiler also adjusts the size of mixing operations to improve performance: prior work has shown that allocating more space to each mixing operation reduces its latency [65]; however, doing so reduces the spatial parallelism available to other concurrently scheduled operations. The compiler accounts for all of the aforementioned information, yielding a clear and concise problem formulation that can be solved using either exact or heuristic means.

The paper is organized as follows: § 2 provides an overview of the SP-LoC technology that we target in this paper; § 3 presents the compiler and emphasizes the optimization problems that must be solved, along with their interactions. Sections 4 and 5, respectively present our implementation and simulation-based empirical evaluation, including comparison to prior work. Section 6 summarizes related work on DMFB compilation to put the contribution of this paper in context. Lastly, § 7 concludes the paper and outlines directions for future work.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Background

2.1 Language Design for SP-LoCs

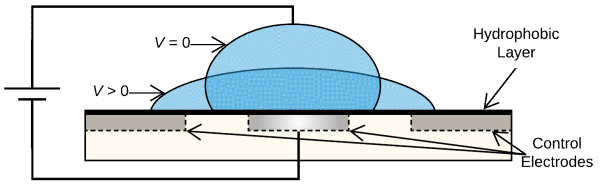
An *assay* is a laboratory procedure that aims to assess the activity of a target entity, called the *analyte*; as an over-generalization, we use the term assay to represent a biochemical "algorithm" that will execute on an SP-LoC. Ideally, the (bio-)chemist of the future will specify an assay using an appropriately designed domain-specific programming language (DSL). A compiler or interpreter will translate the specification into an executable format that will run on the SP-LoC. A number of domain-specific programming languages have been proposed for SP-LoCs [4, 5, 17, 18, 64, 83, 84, 86]; while most of these languages are tied to specific SP-LoC technologies, any DSL compatible with DMFBs (see § 2.2) could be used as a front-end to the compiler presented here.

2.2 Digital Microfluidic Biochips (DMFBs)

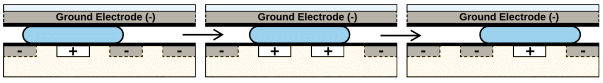
The compiler described in this paper targets a class of SP-LoCs called *Digital Microfluidic Biochips (DMFBs)*, which manipulate discrete droplets of fluid using electrostatic actuation [49, 60]. DMFBs exploit a physical phenomenon called *electrowetting*, shown in Fig. 1a: an electrostatic potential applied to a droplet at rest modifies its shape and angle of contact with the surface; droplet transport can then be achieved by activating and deactivating adjacent electrodes in sequence, as shown in Fig. 1b. An optional top "ground electrode" reduces the voltage required to move a droplet and improves the fidelity of on-chip operations.

A DMFB is a 2D electrode array (Fig. 2a) which supports an instruction set consisting of five operations: store, transport, mix, merge, and split (Fig. 2b) [1, 26, 32, 59, 62, 68]. An "executable program" is a sequence of electrode activations supplied by a host PC or microcontroller. A compiler translates a text-based assay specification into an executable program [18, 64]. A DMFB is "reconfigurable" in the sense that each operation can be performed anywhere on the electrode array and any given electrode may contribute to different operations at different points in time during execution. A typical DMFB will integrate non-reconfigurable resources such as I/O reservoirs on its perimeters, as well as heaters [53] or optical detectors [51, 52, 78, 85] into the array itself. All five basic operations can be performed at the same location as a heater (when off) or a detector; however, heating and detection cannot be performed at any location on-chip. Thus, a compiler must know the precise location of all I/O pads on the device perimeter and both the location and function of all other integrated components; these impose constraints that the compiler's code generator must satisfy.

Integration of sensors [1, 8, 16, 23, 43, 45, 46, 56, 61, 69, 73–76, 82] and online video monitoring [2, 3, 33, 36–39, 47, 54, 55, 66, 93] allows a CPU controlling a DMFB to obtain online feedback regarding the state of the assay during execution.

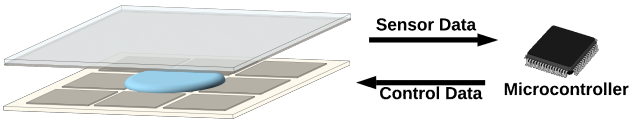


(a) The electrowetting effect: applying an electrostatic potential to a droplet modifies its contact angle [49, 60].

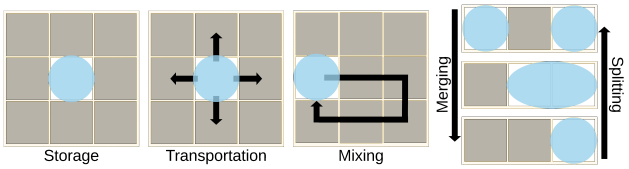


(b) A droplet transport is achieved by activating and deactivating electrodes in sequence.

Figure 1. (a) The electrowetting principle, (b) droplet transport



(a) A DMFB (left) is comprised of a 2D array of discrete electrodes, with an optional ground electrode on top. A cyber-physical feedback loop to a microcontroller is enabled by sensory feedback.



(b) DMFB instruction set architecture (ISA).

Figure 2. (a) A DMFB and (b) its 5-operation ISA.

At the language design level, this provides control flow: arbitrary computations can be performed on acquired sensory data, including predicates that resolve conditions at runtime [18, 31]. The compiler must ensure that all droplets are routed to the same location at the start of each basic block, regardless of which control paths are taken [18].

2.3 Mixing Modules

The latency of mixing two fluids depends on the number of electrodes that have been allocated to perform the mixing and also the routing path that the droplet takes within the mixer [65] (see Table 1). While larger mixers yield lower latency, they reduce the availability of spatial parallelism on-chip. The compiler described here includes a feedback loop that adjusts the size of different mixing operations in order to optimize performance.

Table 1. Mixing module dimensions and their latencies [65].

Size	2 × 2	2 × 3	1 × 4	2 × 4
Mixing time (sec)	9.95	6.61	4.6	2.9

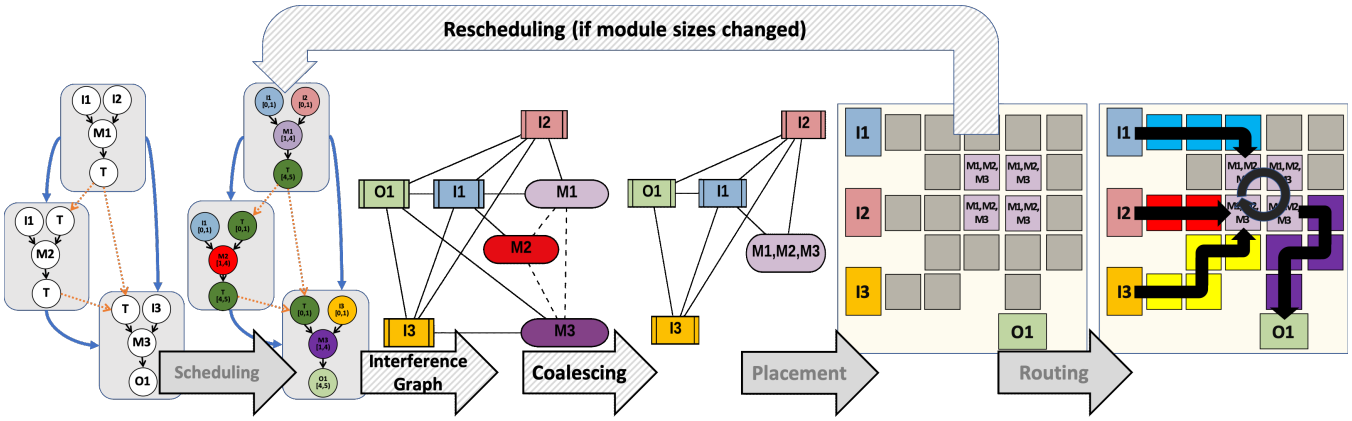


Figure 3. Overview of our DMFB compiler. The front-end compiles an assay specification to a CFG (not shown). The back-end converts the CFG to an executable format. The "Interference Graph", "coalescing", and "rescheduling" arrows are the novel aspects of this paper.

3 Compiler

3.1 Overview

The assay is specified in a domain-specific language such as BioCoder [18] or BioScript [64], that seamlessly interleaves fluidic operations with computation. The target is a cyber-physical DMFB (Fig. 2) which provides sensory feedback to the runtime software that manages the device. This enables the programmer to specify assays featuring arbitrary control flow: the assay obtains sensory feedback from the device and performs computations on the acquired data; the result of the computation can be used as a condition which determines which fluidic operations to execute next.

Our input language supports function calls, but does not support unbounded recursion. The compiler's preprocessor inlines all function calls, which converts the assay to one procedure. The input language restricts all fluidic variables to be scalars; it does not support fluidic arrays. We hope to relax these assumptions in the future.

Figure 4a depicts an assay specified in the BioScript language [64]. We first convert the assay to a hybrid computational-fluidic *intermediate representation (IR)* [18], as shown in Fig. 4b. This IR represents the assay as a *Control Flow Graph (CFG)*. Next, we convert both fluidic and computational variables to *Static Single Information (SSI) Form* [6, 10, 77]; each basic block is represented as a hybrid-fluidic/data dependence graph. Figures 4c and 4d respectively show the BioScript specification and hybrid-IR converted to SSI Form: in this case, π- and ϕ- functions[1] are inserted for one fluidic variable. Figure 3 outlines the subsequent steps of the compiler's back-end. The following subsections discuss each step in greater detail.

[1]SSI's π-function (sometimes σ-function) defines a split set for a variable at the end of some basic blocks where control flow follows, allowing a unique identifier for each conditional usage of the variable in a similar way that a ϕ-function provides a single definition point for each variable.

3.2 Scheduling

The first step is to schedule assay operations. Each basic block is scheduled individually. The scheduler ensures that each operation starts and finishes within the basic block containing it to ensure atomicity. Referring back to Table 1, the scheduler assumes 2×2 mixers with 9.95s latencies; this assumption is later relaxed during Rescheduling (§ 3.5.1). O'Neal et al. [63] present the problem formulation and survey many scheduling heuristics that have been published to date.

The compiler infers droplet storage operations from the schedule and inserts them into the IR. The IR treats storage as an explicit operation that uses (and consumes) its input and defines a new output droplet. This may necessitate the insertion of additional π- and ϕ- functions to maintain SSI Form, as shown in Figs. 5a and 5b. This representation enables the placer (§ 3.5) to treat droplet storage the same as all other scheduled assay operations.

The scheduler enforces resource constraints that conservatively over-approximate placement. To simplify the discussion, we omit resource constraints involving I/O operations. The scheduler partitions the DMFB into N modules (Fig. 6). At any point in the schedule, a *reconfigurable module* can perform one mix, split, or merge operation, or can store up to k droplets, depending on its size. Any module that features an integrated heater or sensor can perform a heating or sensing operation as well; let the number of such modules be N_{heat} and N_{sense} respectively. Let $r_j(p)$ be the number of operations of type $j \in \{mix, split, merge, store, heat, sense\}$ scheduled at program point p. A legal schedule must satisfy the following constraints for each program point p:

$$r_{heat}(p) \leq N_{heat} \tag{1}$$

$$r_{sense}(p) \leq N_{sense} \tag{2}$$

$$r_{mix}(p) + r_{split}(p) + r_{merge}(p) + \left\lceil \frac{r_{store}(p)}{k} \right\rceil + r_{heat}(p) + r_{sense}(p) \leq N \tag{3}$$

```
1  dispense a
2  dispense b
3  dispense c
4  heat a
5  heat c
6  d = mix a with b
7  detect d
8  if (...)
9    dispense a
10   d = mix a with d
11 d = mix c with d
12 heat d
13 drain d
```

```
1  dispense a0
2  dispense b0
3  dispense c0
4  a1 = heat a0
5  c1 = heat c0
6  d1 = mix a1 with b0
7  detect d1
8  d2, d3 = π(d1)
9  if (...)
10   dispense a2
11   d4 = mix a2 with d2
12 d5 = φ(d3, d4)
13 d6 = mix c1 with d5
14 d7 = heat d6
15 drain d7
```

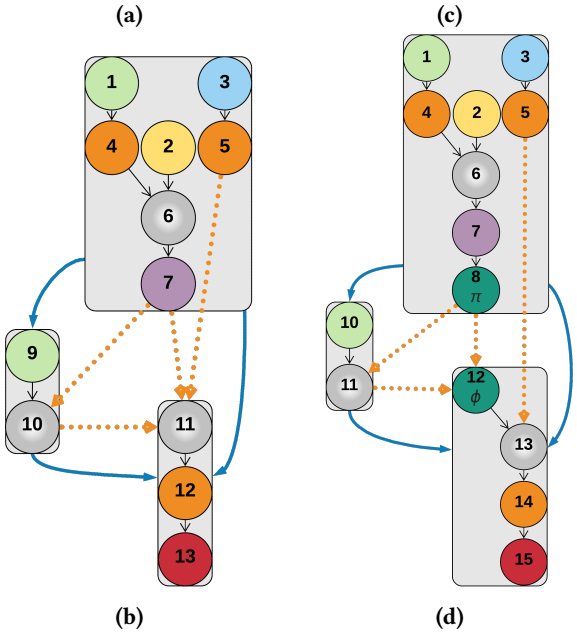


Figure 4. A simple assay written in BioScript (a) and the associated CFG (b) can be augmented with SSI form's ϕ and π nodes (c and d).

Scheduling failures may occur and are unavoidable in the general case, even if the problem is solved optimally. If scheduling fails, the only option is to switch to a larger DFMB target, or rewrite the assay. During compilation, switching to larger and faster mixers Table 1 increases the likelihood of failure, which is one reason why we default to the smallest, slowest mixer for the initial scheduling step. Failures due to module sizes are addressed in § 3.5.1.

3.3 Interference Graph

3.3.1 Definitions and Properties

Let $G = (V, E, A)$ be the *interference graph* [11–13]: V is the set of assay operations, E is the set of interference edges, and A is the set of affinity edges that represent fluid transfers between operations. Let $adj[o_i]$ and $aff[o_i]$ denote the sets of interference and affinity neighbors of $o_i \in V$; additionally, let $adj^*[o_i] = adj[o_i] \cup \{o_i\}$ and $aff^*[o_i] = aff[o_i] \cup \{o_i\}$.

```
1  dispense a0
2  dispense b0
3  dispense c0
4  a1 = heat a0
5  c1 = heat c0
6  d1 = mix a1 with b0
7  detect d1
8  c2 = store c1
9  d2, d3 = π(d1)
10 c3, c4 = π(c2)
11 if (...)
12   dispense a2
13   d4 = mix a2 with d2
14   c5 = store c3
15 d5 = φ(d3, d4)
16 c6 = φ(c4, c5
17 d6 = mix c6 with d5
18 d7 = heat d6
19 drain d7
```

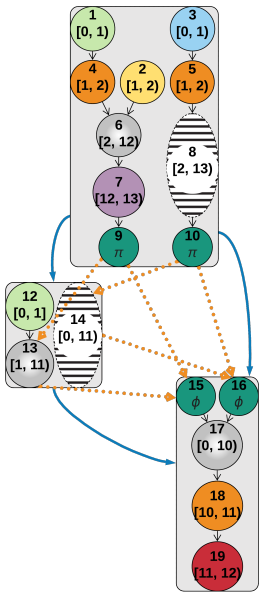


Figure 5. Our scheduler adds implicit store operations (a) and updates SSI form to generate a schedule (b) that captures the linear def-use chain that SSI form provides.

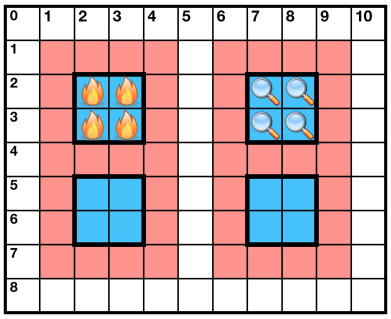


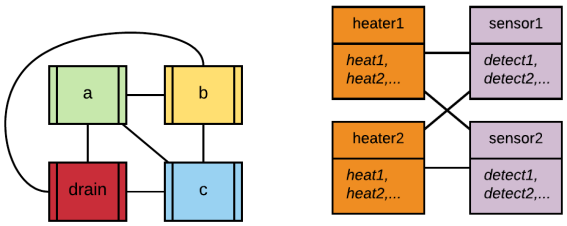
Figure 6. A DMFB partitioned into a 2×2 array of modules exposed to the scheduler: one module has a heater and one has a sensor.

Each vertex is labeled with a *type*, denoted $type[o_i] \in \{mix, split, merge, store, heat, sense\}$. As shorthand, and albeit a slight abuse of notation, we define a *meta-type*, *reconfig*, as the union of types *mix*, *merge*, *split*, or *store*.

The set of interference or affinity neighbors of type t are respectively denoted $adj_t[o_i] = \{o_j \in adj[o_i] \mid type[o_j] = t\}$ and $aff_t[o_i] = \{o_j \in aff[o_i] \mid type[o_j] = t\}$; $adj_t{}^*[o_i]$ and $aff_t{}^*[o_i]$ are defined analogously to $adj^*[o_i]$ and $aff^*[o_i]$.

3.3.2 Construction

An interference edge $(o_i, o_j) \in E$ is placed between two operations o_i and o_j whose lifetimes overlap. Affinity edges arise from fluidic dependencies in the IR, including those arising between fluidic variables used and defined by the ϕ- and π-functions inserted during SSI construction [18, 64].



(a) I/O interferences **(b)** External module interferences

Figure 7. *I/O & Module Interferences:* All I/O reservoirs (a) are universal nodes; their subgraph forms a clique. The subgraph of external modules (b) is a complete multipartite graph, with each module type comprising a part.

An affinity edge $(o_k, o_l) \in A$ indicates that a droplet must be transported between the locations where o_k and o_l are placed. The transport operation can be eliminated if o_k and o_l are placed at the same location.

Affinity edges can only be inserted between "compatible" operations. For example, a mix operation is compatible with a heat because a mix operation can be scheduled on a DMFB module that includes an integrated heater (presumably turned off). On the other hand, heat and sense operations are incompatible: to date no DMFB devices has integrated a heater and sensor at the same on-chip location.

The interference graph includes a complete multipartite gadget (Fig. 7b) to make resource-related incompatibilities explicit. I/O operations bound to the same reservoir cannot interfere, while I/O operations bound to different reservoirs explicitly interfere. Without loss of generality, a sensing operation cannot be bound to a region of a DMFB that features an integrated heater, and vice-versa.

Figure 8a shows the interference graph corresponding to the assay in Fig. 4 after scheduling and storage insertion, and assuming that the target DMFB has at least two heaters. Instructions 1, 2, 3, 9, and 13 are statically bound to I/O reservoirs. Operation 5 (heat c) overlaps with operations 4, 6, and 7; droplet c is stored after operation 7 (detect d) completes. To conserve space, the interference graph omits the interference edges that belong to the gadget in *resource-interferences* between operation 7 (detect d) and the three heat operations (4, 5, and 12). *Fluidic dependencies* result in affinity edges: (v_4, v_6), (v_6, v_7), (v_7, v_{10}), (v_5, v_{11}), (v_7, v_{11}), (v_{10}, v_{11}), and (v_{11}, v_{12}).

3.4 Coalescing

Coalescing merges non-interfering affinity-related vertices in the interference graph to ensure that the corresponding operations are placed at the same on-chip location: this eliminates the need to transport droplets, which can reduce the burden on placement and routing (§§ 3.5 and 3.6), two NP-complete problems. Coalescing is implemented as an affinity edge contraction operation [11, 24, 40, 44]: given an affinity edge $(o_i, o_j) \in A$ where $(o_i, o_j) \notin E$, vertices o_i and o_j are merged to form new vertex o_{ij} having interference and affinity neighbor sets $adj[o_{ij}] = (adj[o_i] \cup adj[o_j])$ and $aff[o_{ij}] = (aff[o_i] \cup aff[o_j]) \setminus \{o_i, o_j\}$. Figure 8a shows the interference graph derived from the scheduled CFG shown in Fig. 5b; Figs. 8b and 8c show two possible coalescing outcomes. In this example, Fig. 8c has coalesced more affinity edges than Fig. 8b. This, in turn, reduces the workload of the placer (§ 3.5) and router (§ 3.6) downstream.

Coalescing here differs from register allocation in one key respect. Consider the example shown in Fig. 9a: when coalescing (o_i, o_j) into o_{ij}, traditional mechanisms discard (o_i, o_k), which may result in extended routes (Fig. 9b). We instead maintain the affinity, allowing routes to be optimized by placing operations *near* each other (Fig. 9c).

When reconfigurable operations of different dimensions are coalesced, the coalesced vertex is given the minimum rectangular dimension that can accommodate its constitutions (see Fig. 10a). The type of a coalesced vertex has the most restrictive among $\{reconfig, heat, sense\}$, as shown in Fig. 10b.

Next, we describe two important subroutines, followed by a description of two coalescing heuristics adapted for our constraints. In the discussion that follows, we talk about interference graph "vertices" rather than assay operations.

Simplification is a subroutine commonly used during register allocation, which we here adapt for our purposes. Any vertex that trivially satisfies the scheduling resource constraints above, but is *not* affinity-adjacent to any other vertices can be removed from the graph: the rationale is that a legal placement for the simplified vertex can *always* be found regardless of where all of its neighboring vertices are placed. Removing simplified vertices from the graph creates opportunities for new coalescing while also rendering other vertices simplifiable. Following repeated rounds of simplification, all vertices in the remaining graph can be placed. Simplified vertices can then be placed by processing them in reverse order of their removal.

Conservative Coalescing Coalescing is *conservative* if the coalesced vertex o_{ij} and its interference neighbors satisfy the scheduler's resource constraints (Eqs. (1) to (3)), i.e.:

$$|adj_{heat}{}^*[o_{ij}]| \leq N_{heat} \tag{4}$$

$$|adj_{sense}{}^*[o_{ij}]| \leq N_{sense} \tag{5}$$

$$|adj_{mix}{}^*[o_{ij}]| + |adj_{split}{}^*[o_{ij}]| + |adj_{merge}{}^*[o_{ij}]| + \left\lceil \frac{|adj_{store}{}^*[o_{ij}]|}{k} \right\rceil + |adj_{heat}{}^*[o_{ij}]| + |adj_{sense}{}^*[o_{ij}]| \leq N \tag{6}$$

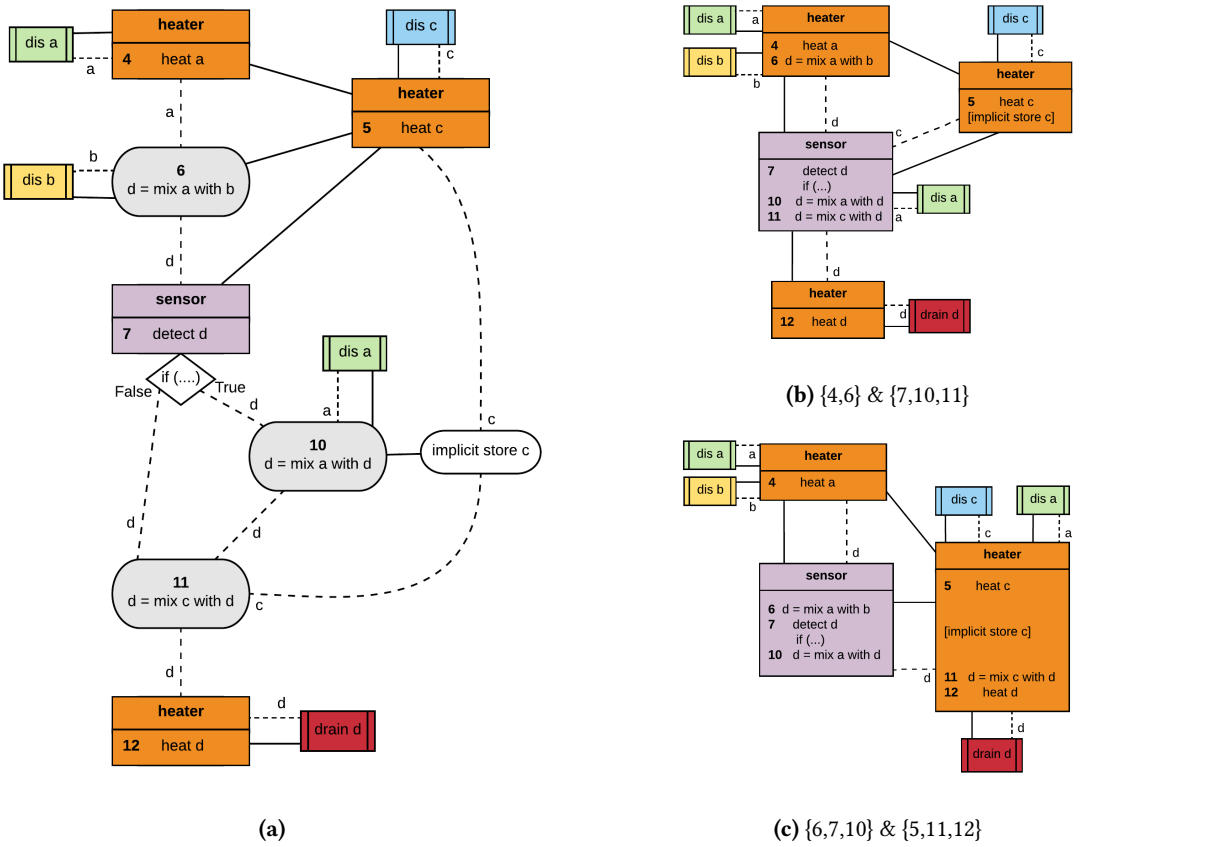


Figure 8. (a) The interference graph for the Assay in Figure 4a: solid lines are interferences, while dotted lines are affinities between nodes. Note that all interferences in Figure 7 are present but not depicted; **(b)** and **(c)** show that a coalescing solution is not unique.

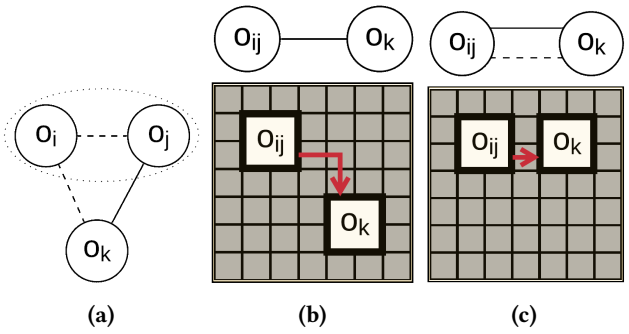


Figure 9. (a) o_i has affinity with o_j and o_k, while o_j and o_k interfere; traditional coalescing does not maintain the affinity edge when coalescing o_{ij}, which may result in extended routes **(b)**; by keeping the edge, we can optimize routes by placing dependent operations *near* each other **(c)**.

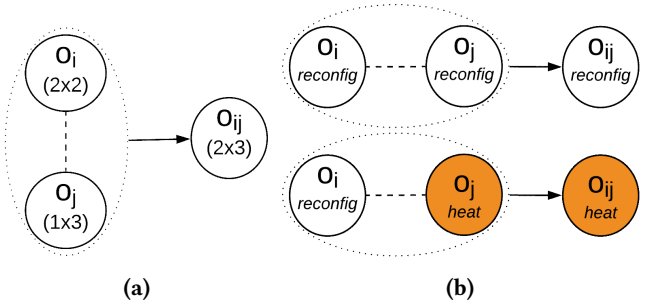


Figure 10. The rectangular dimensions of a coalesced vertex are the minimum dimensions that can accommodate its constituent parts **(a)**; a coalesced vertex takes on the type of its most restrictive module **(b)**.

3.4.1 Coalescing Strategy

Iterated Coalescing, depicted in Fig. 11, is adapted from iterated register coalescing [24], but without spilling. The iterated coalescer simplifies the interference graph until it is not possible to do so any further. It then applies conservative coalescing; if coalescing occurs, further simplification is performed; otherwise, an low-degree vertex with at least one incident affinity edge is "frozen" i.e., the coalescer gives up hope of coalescing its incident affinity edges, thereby allowing the vertex to be simplified. Iterated coalescing terminates when all vertices have been removed via simplification. The graph is then rebuilt and passed to the placer. Conservatism

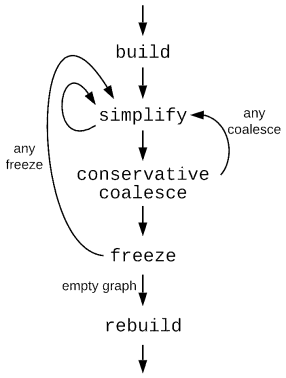


Figure 11. Phase ordering of Iterated Coalescing [24]

is guaranteed by the observation that the initial interference graph, simplification process, and conservative coalescing strategy ensure that the scheduler's resource constraints are satisfied at each step of the heuristic.

3.5 Placement

The placer determines the location on-chip where each assay operation will execute. A legal placement satisfies the constraint that operations o_i and o_j are placed at non-overlapping positions for each interference edge $(o_i, o_j) \in E$. Our compiler implements two distinct placement strategies that have been published elsewhere: ***Virtual Topology with Left-Edge Binder (VT-LEB)*** [28] and ***Keep All Maximal Empty Rectangles (KAMER)*** [7]. Prior work implemented these heuristics in a manner similar to linear scan register allocation [67]. Starting with a scheduled basic block, the placer scans each program point in sequential order: operations scheduled to complete at the previous time-step are removed from the current placement, and operations scheduled to begin at the subsequent time-step are added to the placement.

Our compiler uses modified versions of VT-LEB and KAMER to perform placement on a coalesced interference graph rather than a scheduled CFG; vertices are processed one-by-one in a worklist sorted by the earliest time step.

When coalescing is performed, affinity relationships between interfering vertices may still exist, indicating exactly which vertices should be placed near each other; hence, after placing o_i, if $aff[o_i] \neq \emptyset$, we recursively process affinity neighbors prior to returning to the sorted order (see Fig. 9c).

Let $adj^{<}[o_i]$ be the set of o_i's interference neighbors that precede o_i in the computed order. Placement proceeds in a greedy fashion: operation o_i can be placed at any position that does not overlap the position(s) where operations in $adj^{<}[o_i]$ have been placed. All vertices that have been coalesced with/into o_i are placed at the same location. The resulting placement is guaranteed to be legal as it ensures that o_i's position never overlaps that of any vertices in $adj[o_i]$. VT-LEB guarantees that a legal placement can be found because it ensures that all placement decisions adhere to scheduling resource constraints. Further details regarding the placement heuristics are available in the supplemental materials.

3.5.1 Mix Operation Resizing and Rescheduling

The rescheduling loop in Fig. 3 enables the compiler to adjust the size of mixing operations (Table 1) to reduce assay execution time. The availability of space to accommodate larger mixing operations is not known until placement; on the other hand, the benefits of adjusting the latency of a mixing operation cannot be ascertained without rescheduling, and the updated schedule may change which fluidic variable live ranges overlap, thereby rendering the interference graph invalid. This observation necessitates the rescheduling loop.

The compiler uses a *local search*, which converges to a locally optimal solution, to adjust mixing operation sizes. When placing an interference graph, the first mixing operation or coalesced vertex o_i that contains at least one mix operation invokes Algorithm 1 to select an appropriate mixer size. The heuristic relies on two subroutines:

1. `MaxParallel` applies Dilworth's Theorem [20] to compute the *width*, i.e., the maximum number of operations that *could* be scheduled concurrently, of the basic block that contains o_i; if o_i contains multiple coalesced vertices, `MaxParallel` returns the maximum width of among all of the basic blocks containing them.
2. `CanFit` computes the number of mixing modules of size s that can fit on a given DMFB architecture. Referring back to Fig. 6, `CanFit` is effectively the same subroutine that a scheduler would use to determine the resource constraints of the target chip.

The heuristic first checks if o_i's scheduled module size `CanFit` `MaxParallel` operations. If more parallelism is available than what is currently scheduled, it checks if *smaller* modules `CanFit` more than those currently scheduled, and continues until it finds a module size that `CanFit` up to `MaxParallel` operations.

The heuristic will *increase* o_i's size in two cases:

1. If the chip `CanFit` strictly fewer than `MaxParallel` operations, and the heuristic is unable to increase the number of operations the chip `CanFit` by decreasing o_i's module size, then it *increases* o_i's size as long as it does not further reduce the number of operations that the chip `CanFit`.
2. If o_i's scheduled module size `CanFit` `MaxParallel` operations, then the heuristic *increases* o_i's size to the largest point where `MaxParallel` operations `CanFit` on the chip.

When the size of a mixing operation is updated, the size of any other mixing operations that are coalesced with it are updated as well. If a mixing operation is updated during placement, its latency is scaled as per Eq. (7) and the compiler

loops back to scheduling:

$$t' = t * latency_{new}/latency_{old} \quad (7)$$

For example, if the compiler changes a 10 second `mix` operation's given work module from a 2×3 to a 2×4 module, then the compiler computes the new latency as $t' = 10*2.9/6.1 \approx 4.76$ seconds. The compiler rounds the new latency up to the next millisecond. The termination criteria to continue on to droplet routing is either (1) module sizes are not updated during placement, so rescheduling is unnecessary, or (2) the loop taken during a rescheduling loop failed during scheduling or placement. In the case of (2), we revert to the last legal schedule and placement found. The interested reader can find an example of module resizing in the supplemental materials.

Algorithm 1 Resizing Heuristic

```
1:  function ChooseModuleSize(Block(s) b, Vertex o_i)
2:      current ← o_i.size
3:      choice ← current
4:      max ← MaxParallel(b)
5:      currNum ← CanFit(current)
6:      updated ← False
7:      if max > currNum then
8:          chosenNum ← currNum
9:          smaller ← current
10:         while smaller ≠ smallest do
11:             smaller ← decrease(current)
12:             check ← CanFit(smaller)
13:             if check > chosenNum then
14:                 choice ← smaller
15:                 updated ← True
16:                 chosenNum ← check
17:                 if chosenNum ≥ max then break
18:     if updated = False then
19:         larger ← decrease(current)
20:         check ← CanFit(larger)
21:         while check = currNum OR check ≥ max do
22:             choice ← larger
23:             if larger = largest then break
24:             larger ← increase(choice)
25:             check ← CanFit(larger)
26:     o_i.size ← choice
```

3.6 Droplet Routing

Once a legal placement solution is obtained, each droplet must be routed from its source to its destination; many papers published in the past 15 years have described routing algorithms, and in principle any can be used [9, 15, 34, 41, 42, 71, 72, 81, 91]. The most advanced routers also integrate washing operations to eliminate cross-contamination [35, 89, 92]. The only additional requirement is that droplet routes must be inserted at basic block boundaries; our compiler implements these routes as part of SSI elimination [18].

4 Implementation

Our compiler targets an open-source cycle-accurate DMFB simulator [27, 29]; we modified a back-end that can statically compile CFGs [18], and rely on the simulator to report execution time. We used a collection of benchmarks specified using the BioScript language, which is compatible with the framework's static compilation model [64]. Our compiler uses List Scheduling [28, 80], the VT-LEB [28] and KAMER [7] placers, and a greedy, yet effective, droplet router [28, 71].

Interference graph construction and coalescing are performed after scheduling (Fig. 3). Coalescing is abstracted away from placement so that any existing placement heuristic could be modified easily to operate on a coalesced interference graph. While rescheduling is abstracted away from placement, the resizing operations, by necessity, must be performed during placement, which necessitates a substantial revamp of the heuristic. Our current implementation is only compatible with placers that place operations one-at-a-time in a greedy fashion. A further discussion of the necessary modifications is available for the curious reader as a supplemental material.

5 Evaluation

Even though we support physical chips, the expense associated with their use is prohibitive for evaluation; hence, we evaluate our compiler through simulation-based empirical studies on known real-world assays specified for execution on DMFBs. Specifically, we aim to evaluate the impact of coalescing and mix operation resizing on compilation and assay execution time. All reported averages use the geometric mean over the ratios of each benchmark to avoid providing too much weight to longer- or shorter-running benchmarks [22].

5.1 Experimental Setup

Experiments were performed on a 2.7 GHz Intel®; Core™ i7 processor, 8GB RAM, machine running macOS®. We compare directly against two previously published compilers [18, 64] using an identical 15×19 DMFB architecture. We also report results on 15×15, 12×12 and 8×8 DMFBs to evaluate the impact of our mix operation resizing heuristic.

5.2 Benchmarks

Our evaluation uses a set of DMFB benchmarks that were previously used to evaluate the two compilers that we use as a baseline for comparison. Ref. [18] specified them using a variant of the BioCoder language, which is now deprecated; Ref. [64], as well as this work, uses the replacement BioScript language; a detailed summary of the benchmarks are given in [64]'s supplemental materials.[2]

5.3 Baseline Compilers

The DMFB compilers we compare against do not employ coalescing or mix operation resizing: Ref. [18] compiles a CFG one basic block at a time using the standard VT-LEB algorithm for placement ([28]), eschewing optimizations across basic block boundaries. Ref. [64] employs the NSGA-II [19] metaheuristic for placement. The NSGA-II placer attempts to maximize the number of affinity-adjacent operations that are placed at the same location, as well as affinity-adjacent operations which interfere nearby one another in order to reduce droplet routing paths, but it does not employ coalescing. The runtime of NSGA-II depends on a complex set of parameter values; to get good results, it needs to run much longer than a greedy heuristic such as VT-LEB or KAMER.

5.4 Results and Analysis

Table 2 compares simulated assay execution times previously reported for the two baseline compilers [18, 64] to three configurations of the compiler presented here: VT-LEB placement plus coalescing (VC), KAMER placement plus coalescing (KC), and KAMER placement plus both coalescing and mix operation resizing (KCR). On average, VC, KC, and KCR reduce assay execution time by 1.1%, 1.2%, and 25.0% respectively. These results are not surprising, as assay execution time is known to be dominated by schedule latency, not droplet routing time [81]; as optimizations, coalescing aims to reduce droplet routing overhead while mix operation resizing can lead to shorter schedules. We observed that convergence typically occurs after 2 iterations of rescheduling when resizing is enabled.

The improvements reported for VC and KC over Ref. [64] indicate situations where coalescing turns out to be more effective than the NSGA-II placer; however, NSGA-II may discover different (and possibly better) solutions if the random number seed and other configuration parameters are varied. Future work may extend the NSGA-II placer to utilize a coalesced interference graph; the amount of work required to extend the NSGA-II placer with resizing capabilities (which would entail re-scheduling and re-placing at every perturbation) is prohibitive, so we did not evaluate this option.

The compiler described in Ref. [18] utilizes the same placer as VC, sans coalescing. Adding coalescing capabilities yielded marginal improvements, due to the fact that droplet routing does not dominate total assay execution time.

Mix operation resizing had a more profound impact on total assay execution time than coalescing. Furthermore, Fig. 12 depicts an observed linear correlation between the amount of time an assay is specified for mixing and the percentage decrease we expect to achieve via resizing across DMFBs of varying size. At the smallest size, 8×8 (Figure 12d), resizing allows us to compile several assays that failed to compile successfully without this optimization turned on. Through inspection, we determined that our resizing heuristic was able to avail the minimum required parallelism for these assays by using a 1×4 module size; the default 2×2 mixer did not provide enough room for a legal schedule.

Table 3 provides details into how coalescing impacts the placer's workload and droplet routing time. On average, coalescing reduces the number of operations that are placed by 77%; this, in turn, reduces the amount of work that needs to be done during both placement and routing. In terms of overall performance impact, the VC and KC placers reduced droplet routing times by 9.4% and 8.6% compared to the baseline.

6 Related Work

The majority of work on DMFB compilation targets devices that do not feature sensory feedback or control flow; as such, the scope of compilation was limited to programs that consisted of a single basic block. Discrete formulations of the various compilation stages of scheduling[21, 30, 50, 63, 70, 80], placement [14, 28, 48, 57, 58, 79, 87, 88, 90], and droplet routing [9, 15, 34, 41, 42, 71, 72, 81, 91] were explored, along with wash-droplet [35, 89, 92] to eliminate contamination on the surface of the chip. The compiler described here is a general framework and could implement any of these algorithms.

Early work on DMFB compilation featuring control flow targeted online error detection and recovery for the single basic-block compilation model described above [2, 3, 33, 36–39, 47, 54, 55, 66, 93]. With appropriate extensions to handle CFGs, these techniques could be integrated into the runtime system that executes assays compiled using the techniques described here on a DMFB; it is beyond the scope of this work to design and evaluate such techniques.

This work builds directly on two prior papers that described techniques for DMFB compilers. The first [18] introduced the hybrid computational-fluidic IR used in this paper, and demonstrated how to compile a CFG: each basic block could be compiled individually, with additional droplet routes inserted at control flow edges. These routes ensure that each basic block begins with its incoming droplets at the same position regardless of which control path is taken leading into that basic block. A subsequent paper [64] introduced the BioScript language (which we use here) and represents the first attempt to optimize placement on the granularity of a CFG, as opposed to individual basic blocks; this provided the ability to optimize the additional droplet routes inserted by the earlier compiler [18]; placement relied on an iterative improvement metaheuristic, which ran slowly but generated locally optimal solutions. The contributions of this paper are threefold: (1) coalescing as a placement strategy; (2) faster-running heuristics that can handle placement on the CFG

[2]We discovered a semantic error in the OpiateDetection assay as used in [18, 64] and have adjusted it for correctness. Also, the BioScript specifications we use from [64]'s supplemental materials do not match results when running the framework. Timing specifications were updated through to match previous results prior to conducting our experiments.

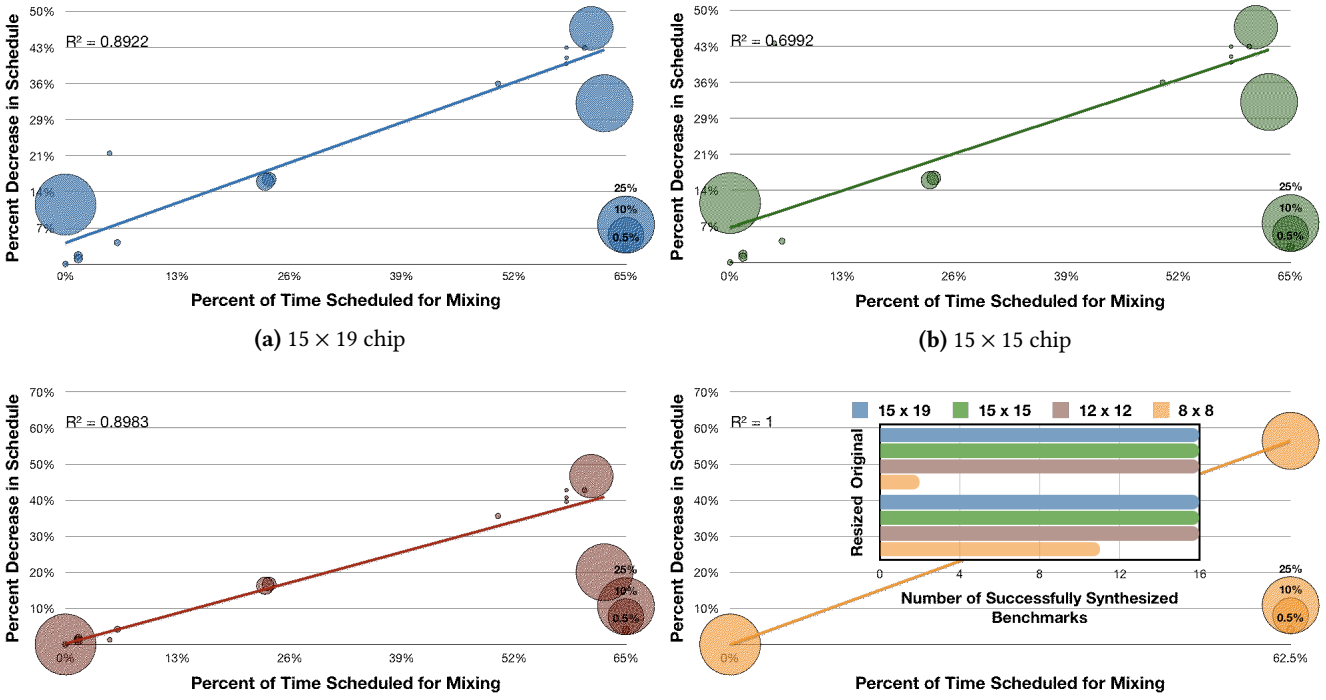
Table 2. Impact of coalescing, choice of placement heuristic, and mix operation resizing on total assay execution time.

	Assay	Total Execution Time (m:s.ms) Baseline	VT-LEB + Coalesce	KAMER + Coalesce	KAMER + Coalesce + Resize
Ott et al., [64]	BroadSpectrumOpiate	00:18.550	00:18.200	00:17.810	00:16.180
	CancerDetection	1920:08.010	1920:06.000	1920:02.810	1919:24.000
	Ciprofloxacin	101:31.800	100:37.100	100:36.910	100:29.950
	Diazepam	96:48.130	96:50.180	96:49.760	96:14.970
	Dilution	21:05.000	20:43.000	20:41.000	06:25.470
	Fentanyl et al.	126:32.400	126:24.540	126:24.330	72:20.600
	FullMorphine[2]	157:21.540	157:21.500	157:19.890	122:52.780
	GlucoseDetection	00:23.770	00:23.590	00:23.730	00:16.730
	ImageProbeSynth	08:38.960	08:22.860	08:22.780	06:58.860
Curtis et al., [18]	OpiateDetection_N[2]	252:50.400	252:50.100	252:47.500	144:36.540
	OpiateDetection_P_H[2]	227:04.000	227:03.700	227:01.800	137:01.140
	OpiateDetection_P_M[2]	353:20.700	353:20.200	353:17.100	209:13.700
	PCRDropletReplacement	40:44.000	39:17.890	39:17.120	32:55.170
	ProbabilisticPCR_early	07:21.000	07:12.420	07:12.390	07:05.430
	ProbabilisticPCR_full	11:19.000	11:10.600	11:10.550	11:03.610
	PCR	11:43.000	11:27.370	11:27.380	11:27.370
	Average Decrease:		*1.1%*	*1.2%*	*25.0%*

N - negative P - positive H - heroin M - morphine

Table 3. Impact of coalescing on placement effort and droplet routing time.

	# Modules Placed		Droplet Route Time (s.ms)		
Assay	Baseline	Coalesced	Baseline	VT-LEB + Coalesce	KAMER + Coalesce
BroadSpectrumOpiate	5	2	00.740	00.740	00.910
CancerDetection	11	4	00.820	00.680	01.000
Ciprofloxacin	11	3	02.390	02.450	02.220
Diazepam	13	2	02.600	02.670	03.950
Dilution	11	2	00.710	00.750	01.150
Fentanyl et al.	11	3	02.670	02.770	02.540
FullMorphine[2]	19	8	05.840	05.760	08.420
GlucoseDetection	10	5	01.470	01.250	01.800
ImageProbeSynth	9	1	00.770	00.530	00.780
OpiateDetection_N[2]	49	4	06.060	05.030	07.230
OpiateDetection_P_H[2]	49	4	05.820	04.770	06.870
OpiateDetection_P_M[2]	49	4	08.470	06.890	10.030
PCRDropletReplacement	4	2	00.510	00.510	00.350
ProbabilisticPCR_early	10	2	07.610	03.920	05.420
ProbabilisticPCR_full	8	2	00.630	00.530	00.390
PCR	8	2	00.870	00.770	00.550
Average Decrease:		*77.1%*		*9.4%*	*8.6%*



(a) 15 × 19 chip

(b) 15 × 15 chip

(c) 12 × 12 chip

(d) 8 × 8 chip is hamstrung without resizing

Figure 12. Resizing mix operations: we observe a linear correlation between the ratio of time spent mixing and the expected percent decrease in an assay's total schedule. The size of the bubbles indicate the ratio of time scheduled for I/O operations. Without resizing, the 8 × 8 chip can only synthesize 2 of the 18 benchmarks. With resizing enabled, we are able to successfully synthesize 11 out of the 18.

granularity; and (3) mixing operation resizing, which has a much greater impact on performance than coalescing.

Another approach, which is orthogonal to what we propose here, is to interpret assays online, rather than compile them offline [31, 86]. The interpreter JIT-compiles each basic block in an on-demand fashion, emphasizing compilation speed over solution quality. To the best of our knowledge, prior work has not attempted to JIT-compile an assay on the granularity of the CFG; any such approach could build on the techniques used here, noting that the runtime overhead of mix operation resizing may be prohibitive. Further, there is a complex interplay between coalescing and module resizing, as resizing may affect interferences across the CFG during rescheduling; hence, the combination of these optimizations are not well-suited for online compilation.

7 Conclusion and Future Work

This paper described the framework of an optimizing compiler for DMFBs; the key innovations were twofold: the formulation of the placement problem for CFGs that shares many principle similarities to register allocation [12, 13], which enabled the adaptation of register coalescing techniques [11, 24] to eliminating otherwise spurious droplet routes, and a mix operation resizing step to reduce schedule latency. While there is certainly room to investigate more effective heuristics that solve the various problems within the compiler, we believe that the general back-end framework presented here represents the correct way to model the constituent optimization problems that must be solved, along with their interactions. Moreover, we believe that the most important topics for future investigation start at the programming language level; for example, determining how to support function calls, fluidic arrays, and fluidic SIMD operations; additionally, there is need to port BioScript (and/or other similar languages) to a variety of SP-LoC targets in addition to DMFBs.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1351115, Grant No. 1536026, Grant No. 1545097, and Grant No. 1910878. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Mirela Alistar and Urs Gaudenz. 2017. OpenDrop: An Integrated Do-It-Yourself Platform for Personal Use of Biochips. *Bioengineering* 4, 2 (2017), 45. https://doi.org/10.3390/bioengineering4020045

[2] Mirela Alistar and Paul Pop. 2015. Synthesis of biochemical applications on digital microfluidic biochips with operation execution time variability. *Integration* 51 (2015), 158–168. https://doi.org/10.1016/j.vlsi.2015.02.004

[3] Mirela Alistar, Paul Pop, and Jan Madsen. 2016. Synthesis of Application-Specific Fault-Tolerant Digital Microfluidic Biochip Architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems* 35, 5 (2016), 764–777. https://doi.org/10.1109/TCAD.2016.2528498

[4] Ahmed M. Amin, Raviraj Thakur, Seth Madren, Han-Sheng Chuang, Mithuna Thottethodi, T. N. Vijaykumar, Steven T. Wereley, and Stephen C. Jacobson. 2013. Software-programmable continuous-flow multi-purpose lab-on-a-chip. *Microfluid Nanofluidics* 15, 5 (Nov 2013), 647–659.

[5] Ahmed M. Amin, Mithuna Thottethodi, T. N. Vijaykumar, Steven Wereley, and Stephen C. Jacobson. 2007. Aquacore: a programmable architecture for microfluidics. In *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, Dean M. Tullsen and Brad Calder (Eds.). ACM, New York, NY, USA, 254–265. https://doi.org/10.1145/1250662.1250694

[6] Scott C. Ananian and Arthur C. Smith. 1999. *The Static Single Information Form.* Ph.D. Dissertation. Massachusetts Institue of Technology.

[7] Kia Bazargan, Ryan Kastner, and Majid Sarrafzadeh. 2000. Fast Template Placement for Reconfigurable Computing Systems. *IEEE Design & Test of Computers* 17 (2000), 68–83. https://doi.org/10.1109/54.825678

[8] Biddut Bhattacharjee and Homayoun Najjaran. 2012. Droplet sensing by measuring the capacitance between coplanar electrodes in a digital microfluidic system. *Lab Chip* 12 (2012), 4416–4423. Issue 21. https://doi.org/10.1039/C2LC40647K

[9] Karl-Friedrich Böhringer. 2006. Modeling and Controlling Parallel Tasks in Droplet-Based Microfluidic Systems. *IEEE Trans. on CAD of Integrated Circuits and Systems* 25, 2 (2006), 334–344. https://doi.org/10.1109/TCAD.2005.855958

[10] Benoit Boissinot, Philip Brisk, Alain Darte, and Fabrice Rastello. 2012. SSI Properties Revisited. *ACM Trans. Embedded Comput. Syst.* 11, S1 (2012), 21. https://doi.org/10.1145/2180887.2180898

[11] Preston Briggs, Keith D Cooper, and Linda Torczon. 1994. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 3 (1994), 428–455.

[12] Gregory J. Chaitin. 1982. Register Allocation & Spilling via Graph Coloring. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, Boston, Massachusetts, USA, June 23-25, 1982*, John R. White and Frances E. Allen (Eds.). ACM, Boston, MA, 98–105. https://doi.org/10.1145/800230.806984

[13] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. 1981. Register Allocation Via Coloring. *Comput. Lang.* 6, 1 (1981), 47–57. https://doi.org/10.1016/0096-0551(81)90048-5

[14] Ying-Han Chen, Chung-Lun Hsu, Li-Chen Tsai, Tsung-Wei Huang, and Tsung-Yi Ho. 2013. A Reliability-Oriented Placement Algorithm for Reconfigurable Digital Microfluidic Biochips Using 3-D Deferred Decision Making Technique. *IEEE Trans. on CAD of Integrated Circuits and Systems* 32, 8 (2013), 1151–1162. https://doi.org/10.1109/TCAD.2013.2249558

[15] Minsik Cho and David Z. Pan. 2008. A High-Performance Droplet Routing Algorithm for Digital Microfluidic Biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems* 27, 10 (2008), 1714–1724. https://doi.org/10.1109/TCAD.2008.2003282

[16] Peter Cooreman, Ronald Thoelen, Jean Manca, M. vandeVen, V. Vermeeren, L. Michiels, M. Ameloot, and P. Wagner. 2005. Impedimetric immunosensors based on the conjugated polymer PPV. *Biosens. Bioelectron.* 20 (2005), 2151–2156. Issue 10.

[17] Christopher Curtis and Philip Brisk. 2015. Simulation of feedback-driven PCR assays on a 2D electrowetting array using a domain-specific high-level biological programming language. *Microelectronic Engineering* 148 (2015), 110–116.

[18] Christopher Curtis, Daniel T. Grissom, and Philip Brisk. 2018. A compiler for cyber-physical digital microfluidic biochips. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, Vösendorf / Vienna, Austria, February 24-28, 2018*, Jens Knoop,

Markus Schordan, Teresa Johnson, and Michael F. P. O'Boyle (Eds.). ACM, New York, NY, USA, 365–377. https://doi.org/10.1145/3168826

[19] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (Apr 2002), 182–197. https://doi.org/10.1109/4235.996017

[20] R. P. Dilworth. 1950. A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics* 51, 1 (1950), 161–166. http://www.jstor.org/stable/1969503

[21] Jie Ding, Krishnendu Chakrabarty, and Richard B. Fair. 2001. Scheduling of microfluidic operations for reconfigurabletwo-dimensional electrowetting arrays. *IEEE Trans. on CAD of Integrated Circuits and Systems* 20, 12 (2001), 1463–1468. https://doi.org/10.1109/43.969439

[22] Philip J. Fleming and John J. Wallace. 1986. How Not To Lie With Statistics: The Correct Way To Summarize Benchmark Results. *Commun. ACM* 29, 3 (1986), 218–221. https://doi.org/10.1145/5666.5673

[23] Jie Gao, Xianming Liu, Tianlan Chen, Pui-In Mak, Yuguang Du, Mang-I Vai, Bingcheng Lin, and Rui P. Martins. 2013. An intelligent digital microfluidic system with fuzzy-enhanced feedback for multi-droplet manipulation. *Lab Chip* 13 (2013), 443–451. Issue 3. https://doi.org/10.1039/C2LC41156C

[24] Lal George and Andrew W. Appel. 1996. Iterated register coalescing. *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '96* 18, 3 (1996), 208–218. https://doi.org/10.1145/237721.237777

[25] Georges G. E. Gielen (Ed.). 2006. *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*. European Design and Automation Association, Leuven, Belgium. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=11014

[26] Jian Gong and Chang-Jin Kim. 2008. Direct-referencing two-dimensional-array digital microfluidics using multilayer printed circuit board. *J. Microelectromech. Syst.* 17 (2008), 257–264. Issue 2.

[27] Daniel Grissom and Philip Brisk. 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 (CODES+ISSS '12)*. ACM, New York, NY, USA, 413–422. https://doi.org/10.1145/2380445.2380510

[28] Daniel Grissom and Philip Brisk. 2014. Fast Online Synthesis of Digital Microfluidic Biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems* 33, 3 (2014), 356–369. https://doi.org/10.1109/TCAD.2013.2290582

[29] Daniel Grissom, Christopher Curtis, Skyler Windh, Calvin Phung, Navin Kumar, Zachary Zimmerman, O'Neal Kenneth, Jeffrey McDaniel, Nick Liao, and Philip Brisk. 2015. An open-source compiler and PCB synthesis tool for digital microfluidic biochips. *Integration, the VLSI Journal* 51 (2015), 169–193.

[30] Daniel T. Grissom and Philip Brisk. 2012. Path scheduling on digital microfluidic biochips. In *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun (Eds.). ACM, New York, NY, USA, 26–35. https://doi.org/10.1145/2228360.2228367

[31] Daniel T. Grissom, Christopher Curtis, and Philip Brisk. 2014. Interpreting Assays with Control Flow on Digital Microfluidic Biochips. *JETC* 10, 3 (2014), 24:1–24:30. https://doi.org/10.1145/2567669

[32] Ben. Hadwen, G. R. Broder, D. Morganti, A. Jacobs, C. Brown, J. R. Hector, Y. Kubota, and H. Morgan. 2012. Programmable large area digital microfluidic array with integrated droplet sensing for bioassays. *Lab Chip* 12, 18 (Sep 2012), 3305–3313.

[33] Yi-Ling Hsieh, Tsung-Yi Ho, and Krishnendu Chakrabarty. 2014. Biochip Synthesis and Dynamic Error Recovery for Sample Preparation Using Digital Microfluidics. *IEEE Trans. on CAD of Integrated Circuits and Systems* 33, 2 (2014), 183–196. https://doi.org/10.1109/TCAD.2013.2284010

[34] Tsung-Wei Huang and Tsung-Yi Ho. 2009. A fast routability- and performance-driven droplet routing algorithm for digital microfluidic biochips. In *27th International Conference on Computer Design, ICCD 2009, Lake Tahoe, CA, USA, October 4-7, 2009*. IEEE Computer Society, New York, NY, USA, 445–450. https://doi.org/10.1109/ICCD.2009.5413119

[35] Tsung-Wei Huang, Chun-Hsien Lin, and Tsung-Yi Ho. 2010. A Contamination Aware Droplet Routing Algorithm for the Synthesis of Digital Microfluidic Biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems* 29, 11 (2010), 1682–1695. https://doi.org/10.1109/TCAD.2010.2062770

[36] Mohamed Ibrahim and Krishnendu Chakrabarty. 2015. Efficient Error Recovery in Cyberphysical Digital-Microfluidic Biochips. *IEEE Trans. Multi-Scale Computing Systems* 1, 1 (2015), 46–58. https://doi.org/10.1109/TMSCS.2015.2478457

[37] Mohamed Ibrahim and Krishnendu Chakrabarty. 2015. Error recovery in digital microfluidics for personalized medicine. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, Wolfgang Nebel and David Atienza (Eds.). ACM, New York, NY, USA, 247–252. http://dl.acm.org/citation.cfm?id=2755807

[38] Mohamed Ibrahim, Krishnendu Chakrabarty, and Kristin Scott. 2017. Synthesis of Cyberphysical Digital-Microfluidic Biochips for Real-Time Quantitative Analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems* 36, 5 (2017), 733–746. https://doi.org/10.1109/TCAD.2016.2600626

[39] Christopher Jaress, Philip Brisk, and Daniel T. Grissom. 2015. Rapid online fault recovery for cyber-physical digital microfluidic biochips. In *33rd IEEE VLSI Test Symposium, VTS 2015, Napa, CA, USA, April 27-29, 2015*. IEEE Computer Society, New York, NY, USA, 1–6. https://doi.org/10.1109/VTS.2015.7116246

[40] Jinpyo Park and Soo-Mook Moon. 2004. Optimistic register coalescing. *Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.98EX192)* 26, 4 (2004), 196–204. https://doi.org/10.1109/PACT.1998.727246

[41] Oliver Keszöcze, Robert Wille, Krishnendu Chakrabarty, and Rolf Drechsler. 2015. A General and Exact Routing Methodology for Digital Microfluidic Biochips. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015*, Diana Marculescu and Frank Liu (Eds.). IEEE, New York, NY, USA, 874–881. https://doi.org/10.1109/ICCAD.2015.7372663

[42] Oliver Keszöcze, Robert Wille, and Rolf Drechsler. 2014. Exact routing for digital microfluidic biochips with temporary blockages. In *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2014, San Jose, CA, USA, November 3-6, 2014*, Yao-Wen Chang (Ed.). IEEE, New York, NY, USA, 405–410. https://doi.org/10.1109/ICCAD.2014.7001383

[43] Thomas Lederer, Stefan Clara, Bernhard Jakoby, and Wolfgang Hilber. 2012. Integration of impedance spectroscopy sensors in a digital microfluidic platform. *Microsystem Technologies* 18, 7 (01 Aug 2012), 1163–1180. https://doi.org/10.1007/s00542-012-1464-6

[44] Allen Leung and Lal George. 1998. A New MLRISC Register Allocator.

[45] Yiyan Li, Hongzhong Li, and R. Jacob Baker. 2014. Volume and concentration identification by using an electrowetting on dielectric device, In 2014 IEEE Dallas Circuits and Systems Conference (DCAS). *IEEE DCAS* 1, 1, 1–4.

[46] Yiyan Li, Hongzhong Li, and R. Jacob Baker. 2015. A Low-Cost and High-Resolution Droplet Position Detector for an Intelligent Electrowetting on Dielectric Device. *Journal of Laboratory Automation* 20, 6 (2015), 663–669. https://doi.org/10.1177/2211068214566940 arXiv:https://doi.org/10.1177/2211068214566940 PMID: 25609255.

[47] Zipeng Li, Kelvin Yi-Tse Lai, John McCrone, Po-Hsien Yu, Krishnendu Chakrabarty, Miroslav Pajic, Tsung-Yi Ho, and Chen-Yi Lee. 2018. Efficient and Adaptive Error Recovery in a Micro-Electrode-Dot-Array Digital Microfluidic Biochip. *IEEE Trans. on CAD of Integrated Circuits*

and Systems 37, 3 (2018), 601–614. https://doi.org/10.1109/TCAD.2017.2729347

[48] Chen Liao and Shiyan Hu. 2011. Multiscale variation-aware techniques for high-performance digital microfluidic lab-on-a-chip component placement. *IEEE Trans Nanobioscience* 10, 1 (Mar 2011), 51–58.

[49] Gabriel Lippmann. 1875. *Relations entre les phénomènes électriques et capillaires.* Ph.D. Dissertation. Gauthier-Villars Paris, France:.

[50] Chia-Hung Liu, Kuang-Cheng Liu, and Juinn-Dar Huang. 2013. Latency-optimization synthesis with module selection for digital microfluidic biochips. In *2013 IEEE International SOC Conference, Erlangen, Germany, September 4-6, 2013*, Norbert Schuhmann, Kaijian Shi, and Nagi Naganathan (Eds.). IEEE, New York, NY, USA, 159–164. https://doi.org/10.1109/SOCC.2013.6749681

[51] L. Luan, R.D. Evans, N.M. Jokerst, and R.B. Fair. 2008. Integrated optical sensor in a digital microfluidic platform. *IEEE Sensors* 8 (2008), 628–635. Issue 5.

[52] Lin Luan, Matthew W Royal, Randall Evans, Richard B Fair, and Nan M Jokerst. 2012. Chip scale optical microresonator sensors integrated with embedded thin film photodetectors on electrowetting digital microfluidics platforms. *IEEE Sensors Journal* 12, 6 (2012), 1794–1800.

[53] Yan Luo, Bhargab B. Bhattacharya, Tsung-Yi Ho, and Krishnendu Chakrabarty. 2015. Design and Optimization of a Cyberphysical Digital-Microfluidic Biochip for the Polymerase Chain Reaction. *IEEE Trans. on CAD of Integrated Circuits and Systems* 34, 1 (2015), 29–42. https://doi.org/10.1109/TCAD.2014.2363396

[54] Yan Luo, Krishnendu Chakrabarty, and Tsung-Yi Ho. 2013. Error Recovery in Cyberphysical Digital Microfluidic Biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems* 32, 1 (2013), 59–72. https://doi.org/10.1109/TCAD.2012.2211104

[55] Yan Luo, Krishnendu Chakrabarty, and Tsung-Yi Ho. 2013. Real-Time Error Recovery in Cyberphysical Digital-Microfluidic Biochips Using a Compact Dictionary. *IEEE Trans. on CAD of Integrated Circuits and Systems* 32, 12 (2013), 1839–1852. https://doi.org/10.1109/TCAD.2013.2277980

[56] J Ross Macdonald and E Barsoukov. 2005. Impedance spectroscopy: theory, experiment, and applications. *History* 1, 8 (2005), 1–13.

[57] Elena Maftei, Paul Pop, and Jan Madsen. 2010. Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices. *Design Autom. for Emb. Sys.* 14, 3 (2010), 287–307. https://doi.org/10.1007/s10617-010-9059-x

[58] Elena Maftei, Paul Pop, and Jan Madsen. 2013. Module-Based Synthesis of Digital Microfluidic Biochips with Droplet-Aware Operation Execution. *JETC* 9, 1 (2013), 2. https://doi.org/10.1145/2422094.2422096

[59] Hyejin Moon, Sung Kwon. Cho, Robin L. Garrell, and Chang-Jin Kim. 2002. Low voltage electrowetting-on-dielectric. *J. Appl. Phys.* 92 (2002), 4080–4087. Issue 7.

[60] Frieder Mugele and Jeanchristophe Baret. 2005. Electrowetting: from basics to applications. *Journal of Physics: Condensed Matter* 17 (2005), R705–R774.

[61] Miguel Angel Murran and Homayoun Najjaran. 2012. Capacitance-based droplet position estimator for digital microfluidic devices. *Lab Chip* 12 (2012), 2053–2059. Issue 11. https://doi.org/10.1039/C2LC21241B

[62] Joo Hyon Noh, Jiyong Noh, Eric Kreit, Jason Heikenfeld, and Philip D. Rack. 2012. Toward active-matrix lab-on-a-chip: programmable electrofluidic control enabled by arrayed oxide thin film transistors. *Lab Chip* 12, 2 (Jan 2012), 353–360.

[63] Kenneth O'Neal, Daniel T. Grissom, and Philip Brisk. 2018. Resource-Constrained Scheduling for Digital Microfluidic Biochips. *JETC* 14, 1 (2018), 7:1–7:26. https://doi.org/10.1145/3093930

[64] Jason Ott, Tyson Loveless, Chris Curtis, Mohsen Lesani, and Philip Brisk. 2018. BioScript: programming safe chemistry on laboratories-on-a-chip. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 128.

[65] Phil Paik, Vamsee K Pamula, and Richard B Fair. 2003. Rapid droplet mixers for digital microfluidic systems. *Lab on a Chip* 3, 4 (2003), 253–259.

[66] Sudip Poddar, Sarmishtha Ghoshal, Krishnendu Chakrabarty, and Bhargab B. Bhattacharya. 2016. Error-Correcting Sample Preparation with Cyberphysical Digital Microfluidic Lab-on-Chip. *ACM Trans. Design Autom. Electr. Syst.* 22, 1 (2016), 2:1–2:29. https://doi.org/10.1145/2898999

[67] Massimiliano Poletto and Vivek Sarkar. 1999. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.* 21, 5 (1999), 895–913. https://doi.org/10.1145/330249.330250

[68] Michael G. Pollack, Alexander D. Shenderov, and Richard B. Fair. 2002. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab on a Chip* 2, 2 (2002), 96–101.

[69] Hong Ren, Richard B Fair, and Micheal G Pollack. 2004. Automated on-chip droplet dispensing with volume control by electro-wetting actuation and capacitance metering. *Sensors and Actuators B: Chemical* 98, 2-3 (2004), 319–327.

[70] Andrew J. Ricketts, Kevin M. Irick, Narayanan Vijaykrishnan, and Mary Jane Irwin. 2006. Priority scheduling in digital microfluidics-based biochips, See [25], 329–334. https://doi.org/10.1109/DATE.2006.244178

[71] Pranab Roy, Hafizur Rahaman, and Parthasarathi Dasgupta. 2010. A novel droplet routing algorithm for digital microfluidic biochips. In *Proceedings of the 20th ACM Great Lakes Symposium on VLSI 2009, Providence, Rhode Island, USA, May 16-18 2010*, R. Iris Bahar, Fabrizio Lombardi, David Atienza, and Erik Brunvand (Eds.). ACM, New York, NY, USA, 441–446. https://doi.org/10.1145/1785481.1785583

[72] Pranab Roy, Hafizur Rahaman, and Parthasarathi Dasgupta. 2012. Two-level clustering-based techniques for intelligent droplet routing in digital microfluidic biochips. *Integration* 45, 3 (2012), 316–330. https://doi.org/10.1016/j.vlsi.2011.11.006

[73] Saman Sadeghi, Huijiang Ding, Gaurav J. Shah, Supin Chen, Pei Yuin Keng, Chang-Jin "CJ" Kim, and R. Michael van Dam. 2012. On Chip Droplet Characterization: A Practical, High-Sensitivity Measurement of Droplet Impedance in Digital Microfluidics. *Analytical Chemistry* 84, 4 (2012), 1915–1923. https://doi.org/10.1021/ac202715f arXiv:http://dx.doi.org/10.1021/ac202715f PMID: 22248060.

[74] Michael J Schertzer, R Ben Mrad, and Pierre E Sullivan. 2012. Automated detection of particle concentration and chemical reactions in EWOD devices. *Sensors and Actuators B: Chemical* 164, 1 (2012), 1–6.

[75] Steve C. Shih, Irena Barbulovic-Nad, Xuning Yang, Ryan Fobel, and Aaron R. Wheeler. 2013. Digital microfluidics with impedance sensing for integrated cell culture and analysis. *Biosens Bioelectron* 42 (Apr 2013), 314–320.

[76] Steve C. Shih, Ryan Fobel, Paresh Kumar, and Aaron R. Wheeler. 2011. A feedback control system for high-fidelity digital microfluidics. *Lab Chip* 11 (2011), 535–540. Issue 3. https://doi.org/10.1039/C0LC00223B

[77] Jeremy Singer. 2005. *Static Program Analysis based on Virtual Register Renaming.* Ph.D. Dissertation. University of Cambridge, UK.

[78] Vijay Srinivasan, Vamsee Pamula, and Richard Fair. 2004. Droplet-based microfluidic lab-on-a-chip for glucose detection. *Analytica Chimica Acta* 507 (04 2004), 145–150.

[79] Fei Su and Krishnendu Chakrabarty. 2006. Module placement for fault-tolerant microfluidics-based biochips. *ACM Trans. Design Autom. Electr. Syst.* 11, 3 (2006), 682–710. https://doi.org/10.1145/1142980.1142987

[80] Fei Su and Krishnendu Chakrabarty. 2008. High-level synthesis of digital microfluidic biochips. *JETC* 3, 4 (2008), 1. https://doi.org/10.1145/1324177.1324178

[81] Fei Su, William L. Hwang, and Krishnendu Chakrabarty. 2006. Droplet routing in the synthesis of digital microfluidic biochips, See [25], 323–328. https://doi.org/10.1109/DATE.2006.244177

[82] Ian I. Suni. 2008. Impedance methods for electrochemical sensors using nanomaterials. *TrAC Trends in Analytical Chemistry* 27, 7 (2008), 604 – 611. https://doi.org/10.1016/j.trac.2008.03.012 Electroanalysis

Based on Nanomaterials.

[83] William Thies, John Paul Urbanski, Todd Thorsen, and Saman Amarasinghe. 2007. Abstraction layers for scalable microfluidic biocomputing. *Natural Computing* 7, 2 (5 2007), 255–275.

[84] John Paul Urbanski, William Thies, Christopher Rhodes, Saman Amarasinghe, and Todd Thorsen. 2006. Digital microfluidics using soft lithography. *Lab Chip* 6 (2006), 96–104. Issue 1. https://doi.org/10.1039/B510127A

[85] Matthew White Royal, Nan M. Jokerst, and Richard Fair. 2013. Droplet-Based Sensing: Optical Microresonator Sensors Embedded in Digital Electrowetting Microfluidics Systems. *IEEE Sensors Journal* 13 (12 2013), 4733–4742.

[86] Max Willsey, Ashley P. Stephenson, Chris Takahashi, Pranav Vaid, Bichlien H. Nguyen, Michal Piszczek, Christine Betts, Sharon Newman, Sarang Joshi, Karin Strauss, and Luis Ceze. 2019. Puddle: A Dynamic, Error-Correcting, Full-Stack Microfluidics Platform. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM, New York, NY, USA, 183–197. https://doi.org/10.1145/3297858.3304027

[87] Tao Xu and Krishnendu Chakrabarty. 2008. Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips. *JETC* 4, 3 (2008), 11. https://doi.org/10.1145/1389089.1389091

[88] Tao Xu, Krishnendu Chakrabarty, and Fei Su. 2008. Defect-Aware High-Level Synthesis and Module Placement for Microfluidic Biochips. *IEEE Trans. Biomed. Circuits and Systems* 2, 1 (2008), 50–62. https://doi.org/10.1109/TBCAS.2008.918283

[89] Hailong Yao, Qin Wang, Yiren Shen, Tsung Yi Ho, and Yici Cai. 2016. Integrated Functional and Washing Routing Optimization for Cross-Contamination Removal in Digital Microfluidic Biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 8 (2016), 1283–1296. https://doi.org/10.1109/TCAD.2015.2504397

[90] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. 2007. Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation. *JETC* 3, 3 (2007), 13. https://doi.org/10.1145/1295231.1295234

[91] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. 2008. BioRoute: A Network-Flow-Based Routing Algorithm for the Synthesis of Digital Microfluidic Biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems* 27, 11 (2008), 1928–1941. https://doi.org/10.1109/TCAD.2008.2006140

[92] Yang Zhao and Krishnendu Chakrabarty. 2012. Cross-contamination avoidance for droplet routing in digital microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 6 (2012), 817–830. https://doi.org/10.1109/TCAD.2012.2183369

[93] Yang Zhao, Tao Xu, and Krishnendu Chakrabarty. 2010. Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip. *JETC* 6, 3 (2010), 11:1–11:28. https://doi.org/10.1145/1777401.1777404