A Compiler for Cyber-Physical Digital Microfluidic Biochips

Christopher Curtis
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521, USA

Daniel Grissom
Department of Engineering and Computer Science
Azusa Pacific University
Azusa, CA, 91702, USA

Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA, 92521 USA

Abstract

Programmable microfluidic laboratories-on-a-chip (LoCs) offer automation and miniaturization to the life sciences. This paper updates the BioCoder language and introduces a fully static (offline) compiler which can target Digital Microfluidic Biochips (DMFBs), one type of programmable LoC. The language and runtime leverage sensor integration to execute bio-assays which feature online decision-making based on sensory data. The compiler employs a hybrid intermediate representation (IR) that interleaves fluidic operations with computation on sensor data. The IR extends traditional notions of liveness and interference to fluidic variables and operations to target the DMFB, which has abundant spatial parallelism. The code generator converts the IR into: (1) electrode activation sequences for each basic block in the control flow graph; (2) a set of computations performed on sensor data, which dynamically resolve control flow operations; and (3) electrode activation sequences for control flow transfers. The compiler is validated using a simulator which produces animated videos of bioassay execution.

CCS Concepts •**Hardware→Microelectromechanical systems** • **Software and its engineering→Compilers** • *Applied computing→Life and medical sciences* • Computer systems organization→Special purpose systems

Keywords Digital Microfluidics, Domain-specific language

ACM Reference format:

Christopher Curtis, Daniel Grissom, and Philip Brisk. A Compiler for Cyber-Physical Digital Microfluidic Biochips. In Proceedings of 2018 IEEE/ACM International Symposium on Code Generation and Optimization (CGO'18). ACM, New York, NY, USA, February 24-28, 2018 (CGO'18), 13 pages.
https://doi.org/10.1145/3168826

CGO'18, February 24–28, 2018, Vienna, Austria

978-1-4503-5617-6/18/02. . . $15.00
https://doi.org/10.1145/3168826

1 Introduction

Microfluidics is the science of controlled liquid transport at the microliter scale (and below). One application of microfluidics is fully integrated laboratories-on-a-chip (LoCs), which miniaturize laboratory functions previously carried out in wet laboratories, and reduce the volume of costly biological samples and reagents consumed during a chemical reaction; other benefits include automation and reduced human error, as fluidic actuation is often controlled by a computer interface, as opposed to direct human manipulation within a larger laboratory setting.

Software-programmable LoCs (SP-LoCs), encompassing a wide variety of technologies, have existed for more than 10 years [1-8]. The earliest generation of SP-LoCs accepted commands from a computer controller, but provided no direct feedback; these SP-LoCs could execute assays (biochemical reactions) that were arbitrarily complex in terms of the number of biochemical steps that were performed, but otherwise lacked control flow (i.e., dynamic decision-making).

SP-LoCs can provide online feedback to the computer controller through integrated sensors [9-23] and/or online video monitoring [24-29]; we refer to these SP-LoCs as being "cyber-physical" because the online sensing creates a closed feedback loop. Thus far, cyber-physical capabilities have primarily been used for monitoring (e.g., precise positioning) [12, 15, 18, 23, 25-29], and online error detection, and recovery [29-39].

Cyber-physical integration enables SPLoC programmability. Without sensory feedback, a host PC controlling an SP-LoC can issue commands, but cannot interpret their outcomes. With sensory feedback capabilities, the host PC obtains the ability to process sensory data in real-time and make decisions about which commands to execute next. For all intents and purposes, this is the difference between programming model with and without control flow. In terms of compiler design, the former permits specification of assays limited to one basic block: it may be possible to extract parallelism among fluidic operations, but once a schedule has been computed, the executed sequence of fluidic operations is fully deterministic and is known statically. In contrast, the dynamic decision-making facilitated by cyber-physical integration is naturally encapsulated as a control flow graph (CFG). This paper describes how to statically compile a bioassay specified as a CFG, where fluidic operations and computations on sensory data are interleaved.

1.1 SP-LoC Technology: Digital Microfluidics

A Digital Microfluidic Biochip (DMFB) is an SP-LoC comprising a 2D array of electrodes [1, 40-47] (Fig. 1). A droplet sits between a bottom substrate containing the array of control electrodes and a top plate containing a single ground electrode. Fig. 2. demonstrates droplet transport. On the far left, the droplet is centered over control electrode 2 (CE2), but overlaps with neighboring electrodes CE1 and CE3. As CE3 is activated, a change to the electric field causes a phenomenon known as electrowetting to stretch the droplet over the newly activated electrode [1]. Deactivating CE2 allows the droplet to center on the one remaining activated control electrode (CE3). DMFBs as large as 16,800 electrodes have been reported [47].

The DMFB instruction set consists of five operations: transport, split, merge, mix, and store (Fig. 3), as well as input (dispense) and output (disposal, collect) operations (not shown). Integration of sensors and actuators (e.g., heaters) extends the instruction set, but only at locations on the chip where these devices have been physically placed. From the perspective of programmability and compiler design, the DMFB can be understood as a dynamically reconfigurable spatial computing device [50]; reconfiguration refers to the fact that different (groups of) electrodes can perform different functions (e.g., mixing and storage) at different times during bioassay execution.

At the lowest level of abstraction, a DMFB is programmed by generating a sequence of electrode activations. For example, Fig. 4 shows a sequence that transports two droplets, originally centered on electrodes #1 and #16, toward one another. This sequence can be modeled as a linear synchronous finite state machine, where the system remains in each state for a fixed amount of time (e.g., 10ms), which, at a minimum, must be long enough to transport the droplet from one electrode to its neighbor. This is akin to programming a processor in binary.

The electrode actuation sequence for each operation shown in Fig. 3 is deterministic once the electrode(s) that will perform the operation is(are) known. Given a large bioassay, the job of the compiler is to determine when and where each operation occurs. From there, producing the electrode actuation sequence is trivial, like the translation from assembly code to binary.

1.2 Motivating Example

Fig. 5 depicts an opiate detection immunoassay which is organized as a hierarchical decision tree. The process begins with broad spectrum immunoassays for the opiate and benzodiazepam drug classes. A positive test branches into specific immunoassays to differentiate morphine from heroin and codeine and identify oxycodone [51] and fentanyl [52] with a ciprofloxacin [53] immunoassay serving as a false positive control. If cross-reactivity between drugs is observed, then one can differentiate among them through their kinetic binding parameters. We would like to specify this immunoassay using a high-level language. Each individual test is a basic block, while the internal control flow takes the form of a decision tree. The positive or negative outcome of each test (sensory feedback,) determines the next test (basic block) to execute. This paper describes how to compile the decision tree, including the acquisition and processing of sensory data, and how to statically compile control flow transfers.

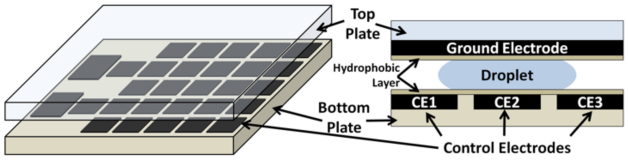

Figure 1. (Left) A Digital Microfluidic Biochip (DMFB) is a 2D array of electrodes [1]. (Right) A cross sectional of a DMFB shows a bottom layer of control electrodes and a top layer containing a ground electrode [48, Fig. 2(a),(b); 49, Fig. 1].

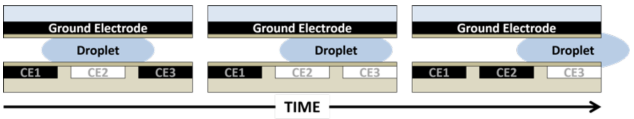

Figure 2. A droplet is moved by activating (white) control electrode 3 (CE3) and deactivating (black) CE2 [49, Fig. 2].

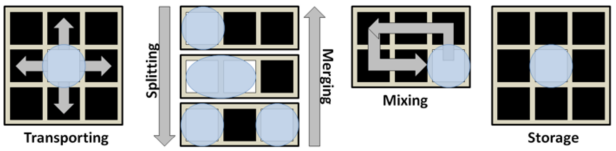

Figure 3. DMFB instruction set: transport, split, merge, mix, and store. Each operation can be executed by an appropriate electrode actuation sequence [48, Fig. 2(c); 49, Fig. 3].

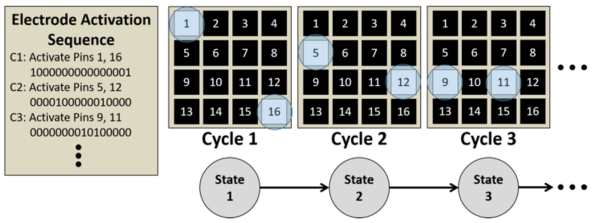

Figure 4. Droplet transport via electrode actuation, similar to machine language (0's and 1's) [48, Fig. 4].

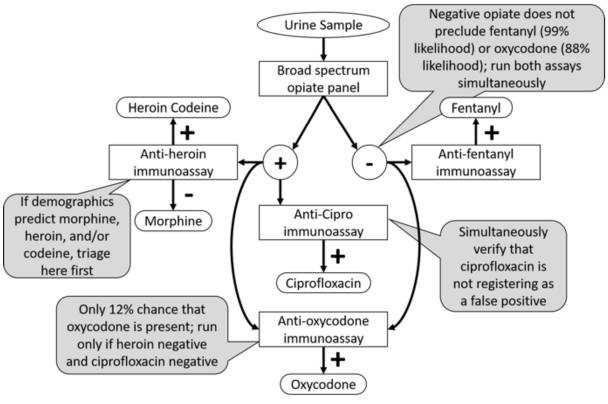

Figure 5. Motivating example: opiate-biased immunoassay organized as a hierarchical decision tree. Feedback occurs at decision points (+ and −), which represent different paths to take if the outcome of each specific test is positive or negative.

Each + or – symbol in Fig. 5 represents a sensor reading which is compared against a control. This notation, which was provided by a collaborator in biomedical engineering, can be converted to conditional statements in a programming language.

Since the aforementioned immunoassays require an antigen to be baked onto the top plate [54], the top plate must either be replaced after each immunoassay is run, or a larger chip must be used where all the enzymes are attached to a single top plate. Replacing the top plate is straightforward and can be done by hand, or autonomously with the help of a robotic arm; these issues are beyond the scope of this paper.

1.3 Contribution

As a proof of concept, we extended BioCoder, a domain-specific language for programmable biochemistry [48, 49, 55], with control flow operations and specialized it to target DMFB technology. We built a compiler for BioCoder, based on the principles outlined in this paper, which targets a DMFB simulator and execution engine that produces animated videos depicting real-time bioassay execution, including real-time decisions based on sensory feedback [56]. This decouples our exploration of fluidic compiler design principles from the formidable challenges associated with obtaining working results on a DMFB in a wet lab, which we leave open for future work.

This paper makes the following contributions: (1) We introduce a new syntax for BioCoder's conditional operations, which improves programmability and readability; (2) we define a hybrid computational-fluidic intermediate representation (IR) which is employed by our compiler; and (3) we describe the steps that the back-end of our compiler performs to convert assays into an executable format appropriate for execution on a DMFB.

```
/*
* restartExperimentDAG drains PCR mix and restarts
* refreshDAG holds replenish PCR Mix Instructions
* finishThermocycleDAG holds thermocycler instrs.
*/

//conditional expression for if
e = new BioExpression(WeightSensor,
                      OP_LT, MinVolumeTolerance);

//if branch "if volume is to low, restart experiment"
bioCond = bcg->
          addNewCondition(e,restartExperimentDAG);
bioCond->addTransferDroplet (InitializationDAGOutputs,
                             restartExperimentDAGInputs);

//else branch "Move to the else if branch"
bioCond = bcg->addNewCondition(NULL, elseIfDAG);
bioCond->addTransferDroplet(InitializationDAGOutputs,
                             elseIfDAGInputs);

//else if branch conditional
e = new BioExpression(WeightSensor,
                      OP_LT, VolumeTolerance);

//else if branch "if volume is low, replenish volume"
bioCond = bcg->addNewCondition(e,refreshDAG);
bioCond->addTransferDroplet(InitializationDAGOutputs,
                             refreshDAGInputs);

//true else branch "Finish the thermocycle"
bioCond = bcg->addNewCondition(NULL,
finishThermocycleDAG);
bioCond->addTransferDroplet(InitializationDAGOutputs,
                             finishThermocycleDAGInputs);
```

2 BioCoder Language

BioCoder is a C/C++ library designed to eliminate ambiguities that occur when biochemical experiments are disseminated in peer-reviewed literature [55] BioCoder was refactored to target microfluidic technologies, including DMFBs [48, 49]. BioCoder sufficed as a proof-of-concept language, but was far too awkward for practical use. For example, fluidic operations, and control flow constructs were specified as programmer-allocated data structures, as shown in Fig. 6, left. This paper updates BioCoder's syntax to introduce the aforementioned statements as language features (Fig. 6, right). The programmer no longer must explicitly allocate conditions and conditional expressions and link them to their targets. The updated syntax supports constant-bounded loops:

```
bioCoder.LOOP(const) ... bioCoder.ENDLOOP()
```

and loops controlled by conditions

```
bioCoder.WHILE(cond) ... bioCoder.ENDWHILE()
```

Programmers can name sensors as variables and use the name in computational expressions and as part of conditions, if desired.

3 Supported Operations

Fig. 7 summarizes the operations supported by the BioCoder language (as specialized for DMFBs) and our compiler. Many of these operations, such as mixing, sensing, and detection are often timed (e.g., "detect for 30s"), and the length of the operation is included explicitly as a constant-valued parameter.

```
bioCoder.IF(WeightSensor, LESS_THAN, MinVolumeTolerance);
    /* Drain current PCR mix and restart experiment
       Replaces restartExperimentDAG */
bioCoder.ELSE_IF(WeightSensor, LESS_THAN,
VolumeTolerance);
    /* Preheat new PCR Mix add it to current mix
       Replaces refreshDAG */
bioCoder.ELSE();
    /* Finish thermocycle instructions.
       Replaces finishThermocycleDAG */
bioCoder.END_IF();
```

Figure 6. An if statement using the original (left) and new (above) BioCoder syntax. In the new syntax, all instructions and fluids are tracked implicitly within the scope of the if statement.

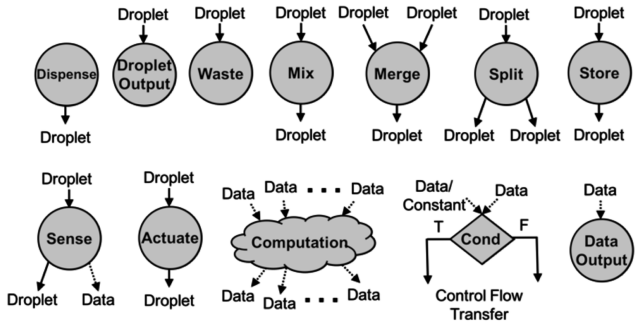

Figure 7. Operations supported by our hybrid computational-fluidic intermediate representation. Dry operations exclusively operate on data (represented by dashed arrows); all other operations are wet and are performed on the DMFB.

Many assays operate on multiple timescales; for example, splitting and merging typically execute on the millisecond timescale, while mixing, sensing, and detection operations may take tens of seconds to complete. Droplet transport and storage operations, depicted earlier in Fig. 3, are not explicitly part of the high-level bioassay specification; the compiler inserts them as needed as part of the back-end optimization process.

Cyber-physical integration of sensors and actuators enables computation to be performed concurrently with assay operations, although at a much faster timescale. As shown in Fig. 8, assay operations are performed on the DMFB, while computation can be offloaded to the host PC controller. In Fig. 7, sensing operations form the link between assay operations and computation. The input to a sensing operation is a droplet; its output is the droplet (usually unmodified) and a scalar data value, which is obtained from the sensor. The computational portion of the assay specification is language-independent and Turing-complete; BioCoder's is based on C/C++ [48, 49, 55] and, in principle, should be used primarily to process sensor data.

In a bioinformatics application, sensor data may be output to the host PC for offline processing, effectively terminating the portion of the bioassay running on the DMFB. Somewhat more generally, we envision that the output of the data processing is the online resolution of a conditional operation that affects program control flow. This imposes the property that the only operations reachable from a data edge are either (1) computations; (2) conditions (noting that the T/F result may be treated as subsequent data edges if desired), or (3) data output.

The sense and actuate operations depicted in Fig. 7 are generic for the purpose of discussion. In actuality, appropriate naming would be used, e.g., "Colorimetric Detection" or "Heat/Cool;" moreover, DMFBs may integrate different types of sensors and/or actuators, each of which will have its own name. When new sensor and actuator technologies are introduced in the future, it will be necessary to update the BioCoder language, intermediate representation, and runtime execution engine to accommodate them; thus, the system has been designed for extensibility and is expected to continuously evolve over time as new technological advances are introduced at the device-level.

Droplets cannot be replicated via copies due to conservation of matter. A droplet copy operation $d' \leftarrow d$ could be used to rename a droplet from d to d', but doing so would effectively kill the original name d. If a second copy of droplet d is needed, then either: (1) the program slice that computed d must be replicated; or (2) the assay must be rewritten to compute d with twice its volume, after which it can be split into d and d'.

4 Preliminaries

The front-end of our compiler converts a BioCoder assay specification into a Control Flow Graph (CFG), which we denote $G_{CFG} = (a, z, B, E_{CFG})$: B is the set of basic blocks, $a, z \in B$ are the unique entry and exit nodes, and $E_{CFG} \subset B \times B$ is the set of directed control flow edges; every basic block $b_i \in B$ exists on at least one path from a to z in G_{CFG}. Each basic block b_i can be represented by a DAG $G_{b_i} = (V_{b_i}, E_{b_i})$; the subscript is dropped for assays consisting of one basic block b without control flow.

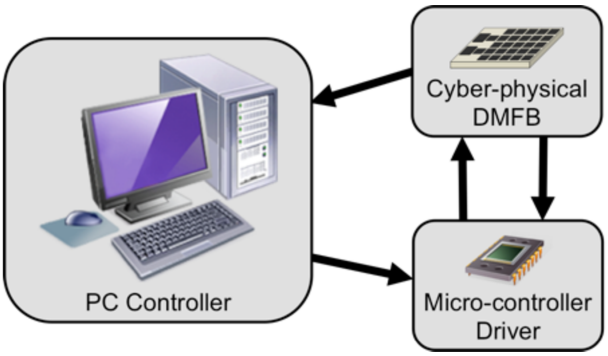

Figure 8. Cyber-physical integration provides a closed feedback loop between the DMFB and a host PC controller. In our system, a microcontroller interfaces directly with the DMFB [49, Fig. 4].

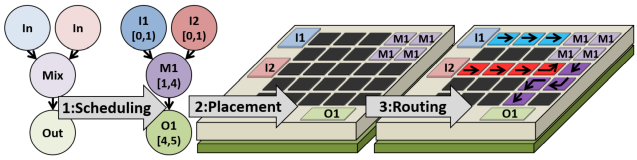

Figure 9. The back-end of a DMFB compiler for a single basic block must solve three interdependent NP-complete problems: scheduling, placement, and routing [48, Fig. 3].

Consider an $M \times N$ DMFB, which is represented by a 2D array of electrodes $\Psi = \{\psi_{i,j} | 1 \le i \le M, 1 \le j \le N\}$; where each electrode is a Boolean $\psi_{i,j} \rightarrow \{0,1\}$, where 0/1 indicates whether or not the electrode is activated. An electrode activation sequence of length T is represented the set $\Sigma = \{\Psi_k | 1 \le k \le T\}$, where Ψ_k denotes the state of each electrode in the array at time k.

The compiler produces an executable $\Delta_{G_{CFG}} = \{\Delta_B, \Delta_{E_{CFG}}\}$, where $\Delta_B = \{\Sigma_{b_i} | 1 \le i \le |B|\}$ is the set of activation sequences for each basic block, and $\Delta_E = \{\Sigma_{(b_i,b_j)} | (b_i, b_j) \in E_{CFG}\}$ is the set of sequences for each control flow edge $(b_i, b_j) \in E_{CFG}$; Σ_a, Σ_z, $\{\Sigma_{(a,b_j)} | (a, b_j) \in E_{CFG}\}$, and $\{\Sigma_{(b_i,z)} | (b_i, z) \in E_{CFG}\}$ are empty.

As noted earlier, the compiler computes the time and location of each assay operation on the array; generating the electrode activation sequence from that information is deterministic.

5 Compiling a Single Basic Block

We consider the degenerate case of compiling an assay that consists of one basic block (represented as a DAG without control flow), a problem that has been studied extensively by others. In this case, the assay may produce sensory data output for offline processing, but otherwise features no online sensory data processing. Fig. 9 shows the three algorithmic stages of the back-end. This assay dispenses two droplets, mixes them (implicitly merging them), and outputs the resultant droplet.

The compiler must solve three interdependent NP-complete problems, scheduling, placement, and droplet routing, in order to produce the electrode activation sequence. The scheduler

computes the start/finish times for each operation [50, 57-62]; the placer determines the location on the DMFB at which each scheduled operation will execute [63-66]; the router computes paths for each droplet when transported between operations [67-76], and introduces wash droplets to clean residue left behind [77-79]. Techniques have also been proposed to solve several of these problems in conjunction with one another [80-85].

Let $v_i \in V_b$ be an assay operation, with start and finish times $s(v_i)$ and $t(v_i)$ respectively. The scheduler computes $s(v_i)$ and $t(v_i)$, under the assumption that droplet routing times are negligible [58, 67], using a conservative approximation of the available spatial resources on the DMFB, while adhering to fluidic dependence constraints, i.e., for edge $(v_i, v_j) \in E_b$, the schedule must ensure that $t(v_i) \leq s(v_j)$. If $t(v_i) = s(v_j)$, then the droplet produced by v_i is used immediately; otherwise, it must be stored for all time steps between $t(v_i)$ and $s(v_j)$. Our scheduler explicitly inserts storage operations into the DAG, ensuring that $t(v_i) = s(v_j)$, for each DAG edge $(v_i, v_j) \in E_b$,

The placer determines an on-chip location $q(v_i)$ for each assay operation v_i; one unit of unused grid space is required between concurrently placed operations, to prevent inadvertent mixing of fluids. Reconfigurable operations (e.g., mix, store) can be placed anywhere. Non-reconfigurable operations (sensing, heating, I/O) must be placed on regions of the chip that feature sensing and/or actuation devices capable of executing them.

The router inserts fluid transport operations, which are encoded into the electrode activation sequence $\Delta_B = \{\Sigma_b\}$. Consider DAG edge $(v_i, v_j) \in E_b$: If $q(v_i) \neq q(v_j)$, the router computes a path from $q(v_i)$ to $q(v_j)$; otherwise no path is needed. Droplets are routed concurrently and wash operations may be interleaved with routing.

6 Compiling a CFG

Now, we consider the more general case of compiling an assay that is specified as a CFG, rather than a single basic block. A BioCoder programmer declares fluids as variables, which are then defined and used, no different in principle than digital variables in traditional software programming. Fluidic variable lifetimes can span multiple basic blocks. A compiler can build data structures and representations for fluidic variables, such as Def-Use trees and SSA Form [86-88] with no modifications being made to the canonical construction algorithms.

Fig. 10 shows a BioCoder specification of an assay that executes the polymerase chain reaction (PCR), which amplifies DNA, using a weight sensor to detect evaporation of the droplet (not the DNA); when the droplet volume falls beneath a threshold during the thermocycling procedure, a new droplet is brought in to replenish the volume [89]. Fig. 11 shows the assay converted to Static Single Information (SSI) Form [90-92].

6.1 Liveness Analysis for Fluids

Liveness analysis for fluidic variables is no different in principle than liveness analysis as performed by a traditional compiler [93]. We define $LiveIn(b_i)$ and $LiveOut(b_i)$ to be the live-in and live-out sets computed for basic block b_i. We assume that a post-processing pass marks all uses that kill each variable.

6.2 Basic Block Scheduling

Next, we compute a schedule for each basic block. This is essentially the same as the scheduling described in Section 5, with the exception that the scheduler must account for liveness information involving fluidic variables:

If fluidic variable $f_j \in LiveIn(b_i)$, then the scheduler treats the basic block entry point as a pseudo-definition. The scheduler inserts storage operations for f_j until it is used by an operation. Subsequent operations that may use f_j are scheduled normally.

If $f_j \in LiveOut(b_i)$, then the scheduler treats the basic block exit point as a pseudo-use. It inserts storage operations for f_j following the last scheduled definition or use of f_j to the end of the schedule for b_i.

The scheduler does not insert storage operations following any use that kills f_j.

It is simple to add these rules/constraints to existing single basic block/DAG schedulers for DMFBs [50, 57-62].

6.3 Placement

6.3.1 Placement for a Single Program Point

We first consider the placement problem for a program point p in basic block b. Let $V_p = \{v \in V_b | p \in L(v)\}$ be the set of operations live at p. Each operation in $v_i \in V_p$ uses a rectangular $m_i \times n_i$ subset of electrodes on the DMFB. The dimensions of an operation that holds a droplet in-place (e.g., storage, sensing, heating, etc.) depend on the droplet volume. The dimensions of mixing operations can vary in size (e.g., 2×2, 2×3, etc.), based on the observation that mixers with greater dimensions often converge faster when sensing and/or imaging is used to detect mixing completion [94][1]. Split operations are typically 1×3.

A unit-size droplet is slightly larger than one electrode; as shown in Fig. 2, this is necessary to induce droplet transport. As noted earlier, placed operations must maintain a separation of at least one electrode-width (grid cell) between them to prevent inadvertent merging [63]. Let M and N be the length and width of the DMFB. A placement solution assigns a position $q(v_i) = (x_i, y_i)$ to each operation $v_i \in V_p$, representing the upper-left-hand corner of its position; when placed, v_i consumes a subset of cells

$$C_i = \{x_i \ldots x_i + m_i - 1\} \times \{y_i \ldots y_i + n_i - 1\}. \quad (1)$$

A *legal* placement solution satisfies the following constraints:

$$x_i \geq 1, x_i + m_i - 1 \leq M, \forall v_i \in V_P \quad (2)$$
$$y_i \geq 1, y_i + n_i - 1 \leq N, \forall v_i \in V_P \quad (3)$$
$$x_j > x_i + m_i \vee x_i > x_j + m_j \vee \quad (4)$$
$$y_j > y_i + n_i \vee y_i > y_j + n_j \;\; \forall v_i, v_j \in V_P, i \neq j$$

[1] Prior work on DMFB scheduling has accounted for mixers with varying mixing times [61, 81-84]; on the other hand, these approaches have not been reconciled with assay specifications in which mixing operations are timed (e.g., "Mix for 2s"). These specifications are typically non-device specific, and it remains an open question as to when it is acceptable to change the assumptions (e.g., 2s mixing for a 2×3 mixer, 3s mixing for a 2×2 mixer, etc.; it is beyond the scope of this work to attempt to reconcile these issues here.

```
void PCRDropletReplenishement (int TotalThermo) {

    int TotalThermo = 9;
    BioSystem bioCoder;
    Fluid *PCRMix = bioCoder.new_fluid("PCRMasterMix",
                                Volume(MICRO_LITER,10));
    Fluid *Template = bioCoder.new_fluid
         ("Template", Volume(MICRO_LITER,10));
    Container* tube = bioCoder.new_container
         (STERILE_MICROFUGE_TUBE2ML);

    bioCoder.measure_fluid(PCRMix,tube);
    bioCoder.vortex(tube,Time(SECS,1));
    bioCoder.measure_fluid(Template,tube);
    bioCoder.vortex(tube, Time(SECS,1));
    bioCoder.store_for(tube,95,Time(SECS,45));

    bioCoder.LOOP(TotalThermo);
       bioCoder.store_for(tube,95,Time(SECS,20));
       bioCoder.weigh(tube,"weightSensor");
       bioCoder.IF("weightSensor",LESS_THAN, 3.57);
          bioCoder.measure_fluid(PCRMix, tube);
          bioCoder.store_for(tube, 95, Time(SECS,45));
          bioCoder.vortex(tube, Time(SECS,1));
       bioCoder.END_IF();
       bioCoder.store_for(tube,50,Time(SECS,30));
       bioCoder.store_for(tube,68,Time(SECS,45));
    bioCoder.END_LOOP();

    bioCoder.store_for(tube,68,Time(MINS,5));
    bioCoder.drain(tube,"PCR");
    bioCoder.end_protocol();
}
```

Figure 10. BioCoder specification of a bioassay that features user-specified control flow. When the droplet weight falls beneath a threshold, a new droplet is dispensed to replenish the volume [89]. The integer parameters in storage operations represent temperatures, converting them to heating operations.

Constraints (2) and (3) ensure that each operation is placed fully within the $M \times N$ confines of the DMFB. Constraint (4) ensures appropriate spacing between modules, as illustrated by Fig.12(a). Additional constraints may be added, for example, to ensure that placed modules do not block I/O ports, that sensing operations are placed on top of electrodes that feature sensors, etc. Fig. 12(b). depicts a placement violation and a legal placement.

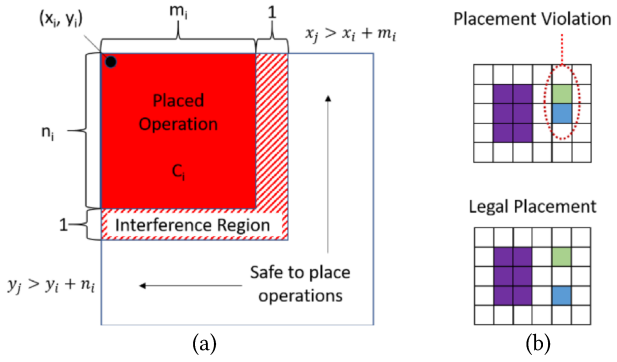

Figure 12. (a) Illustration of placement Constraint (4). (b) the difference between an illegal and a legal placement result.

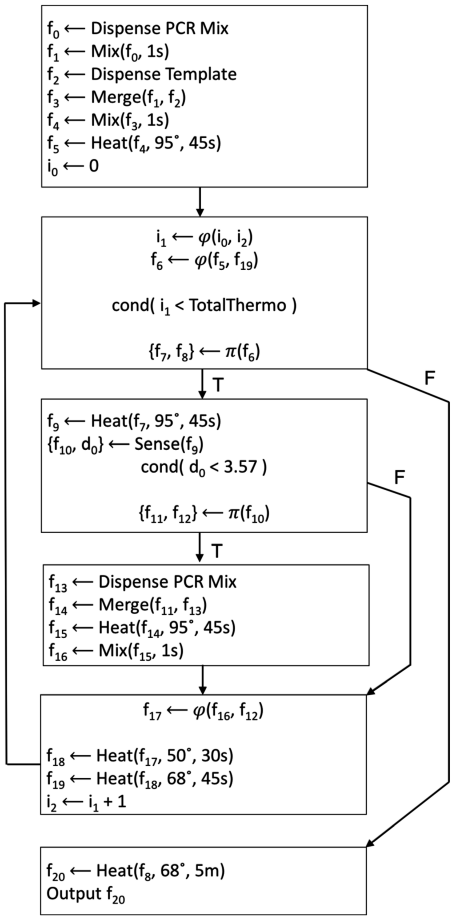

Figure 11. The BioCoder program in Fig. 10 converted to Static Single Information (SSI) Form [90-92].

6.3.2 Basic Block Placement

The placement technique described in the preceding section can be easily extended to a scheduled DAG $G_b = (V_b, E_b)$ for basic block b. The start and finish times of all of the vertices in V_b are sorted in non-decreasing order and processed. Each time a new operation v_i starts, the compiler invokes a greedy heuristic [66, 95] to select a free location on the DMFB to place v_i. Each time an active operation v_j finishes, the compiler reclaims the space that was allocated to it. This approach steps through each program point, as per the schedule of the basic block, and ensures that Constraints (2)-(4) are satisfied.

6.3.3 CFG Placement

Placement for a CFG must account for interferences that cross basic block boundaries; these interferences can only occur for droplet live ranges, which have been converted to storage operations by the scheduler. This necessitates the construction of an interference graph $G_{Int} = (V_{Int}, E_{Int})$, defined as follows.

Let V_{b_i} be the set of scheduled operations in basic block b_i; then $V_{Int} = \bigcup_{i=1}^{|B|} V_{b_i}$.

For operation $v \in V_{Int}$, let $L(v)$ denote the set of points where v is live; then $E_{Int} = \{(v_i, v_j) \in V \mid L(v_i) \cap L(v_j) \neq \emptyset\}$.

Placement for a CFG is similar to placement for a program point, but with Constraint (5) replacing Constraint (4):

$$x_i > x_j + m_j \vee x_j > x_i + m_i \vee \quad (5)$$
$$y_i > y_j + n_j \vee y_j > y_i + n_i \ \forall (v_i, v_j) \in E_{Int}$$

One approach to the algorithmic construction of a placer would be to modify a Chaitin-Briggs style register allocator [96, 97] that eschews spilling and coalescing. When a register allocator would select a "color" for each vertex, a placer could instead call a heuristic to place the corresponding operation.

6.3.4 Placement for a CFG with Live Range Splitting

Our approach to placement for a CFG sidesteps the construction of an interference graph, instead relying on live-range splitting. The basic premise, borrowed from Elementary Form [98], is to split the live ranges of all variables that are live-in to each basic block using φ-functions, and to split the live ranges of all variables that are live-out from each basic block using π-functions, before basic block scheduling (Subsection 6.2). Using this representation, a legal CFG placement can be obtained by placing each basic block independently (Subsection 6.3.2).

6.4 Droplet Routing

Droplet routing is the final algorithmic step that must be performed prior to producing a DMFB executable. Recall that a DMFB executable has the form $\Delta_{G_{CFG}} = \{\Delta_B, \Delta_{E_{CFG}}\}$, where Δ_B is the set of activation sequences for each basic block, and Δ_E is the set of activation sequences for each control flow edge.

6.4.1 Droplet Routing for Basic Blocks

Given a scheduled and placed basic block b_i, a legal droplet routing solution can be achieved using established algorithms [67-76] (with or without washing [77-79]); the electrode activation sequence Σ_{b_i} is computed deterministically. Routing all basic blocks yields $\Delta_B = \{\Sigma_{b_i} | 1 \le i \le |B|\}$.

6.4.2 CFG Placement without Live Range Splitting

Δ_E will be empty if CFG placement is performed without live range splitting, e.g., as described in Subsection 6.3.3. This is because each operation will be placed at the same spatial location on the DMFB at all program points in the CFG. Thus, there is no need to transport any droplets in response to a transfer of control. The case where live range splitting is permitted is discussed next.

6.4.3 Droplet Routing for CFG Edges

Consider a CFG edge (b_i, b_j). Assume that there exists a fluid f such that $f \in LiveOut(b_i)$ and $f \in LiveIn(b_j)$. After inserting φ- and π-functions and scheduling b_i and b_j, assume that f has been renamed to f_k at the end of b_i and is stored in operation v_k, and that f has been renamed to f_l at the beginning of b_j and is stored in operation v_l. After placement, if $q(v_k) = q(v_l)$, there is nothing to do, as v_k/v_l are already in the correct positions. On the other hand, if $q(v_k) \neq q(v_l)$, then a droplet routing procedure must be invoked to transport the droplet from $q(v_k)$ to $q(v_l)$; multiple droplets may need to be transported concurrently. The corresponding electrode activation sequence $\Sigma_{(b_i,b_j)}$ is computed deterministically. Processing all CFG edges yields $\Delta_E = \{\Sigma_{(b_i,b_j)} | (b_i, b_j) \in E_{CFG}\}$.

Fig. 13 shows a few examples, taken from a fragment of the CFG shown in Fig. 11. Fig. 13(a) shows the CFG fragment. Fig. 13(b) illustrates CFG edge (b_1, b_2), which includes a copy operation $f_{11} \leftarrow f_{10}$ via the π-function at the end of b_1. At the end of b_1, f_{10} is placed on an (optical) detector to perform a sensing operation. Since $f \in LiveIn(b_j)$, the scheduler inserts a storage operation for f_{11}; since storage is reconfigurable, this operation could be placed anywhere. We assume that the placer chose to place the storage operation on the sensor, eliminating the need for droplet transport. In this case, it suffices to rename f_{10} to f_{11} in-place $(\Sigma_{(b_1,b_2)} = \phi)$.At the beginning of b_2, a new droplet, f_{13} is dispensed; f_{13} is merged with f_{11}, producing a new droplet f_{14} which is heated. Fig. 13(b) depicts the subsequent droplet transport operations after renaming.

Fig. 13(c) illustrates CFG edge (b_1, b_3). A π-function at the end of b_1 includes a copy operation $f_{12} \leftarrow f_{10}$; the φ-function at the start of b_3 performs a subsequent copy $f_{17} \leftarrow f_{12}$; it suffices to implement a single copy operation $f_{17} \leftarrow f_{10}$. At the end of b_1, f_{10} is placed on a detector, while the first use of f_{17} in b_3 is a heating operation, which necessitates placement on a heater. The droplet router computes a path to transport the droplet from the sensor to the heater; the resulting activation sequence is $\Sigma_{(b_1,b_3)}$.

Fig. 13(d) illustrates CFG edge (b_2, b_3), which includes a copy operation $f_{17} \leftarrow f_{16}$ via the φ-function at the start of b_3. The last use of f_{16} in b_2 is a mixing operation, while the first use of f_{17} in b_3 is a heating operation. In this case, the mixing operation is not placed by the heater, so the droplet router is once again called to compute a path, and the resulting activation sequence is $\Sigma_{(b_2,b_3)}$.

6.4.4 Critical Edge Splitting (or not)

In a traditional software compiler, CFG edges do not contain instructions, by definition. If an instruction needs to be placed on a CFG edge, then the edge must be split, with a basic block inserted, as is the case in the transformation out of SSA Form [93-97]. In contrast, our DMFB executable definition allows the association of electrode activation sequences with CFG edges. This is permissible because the DMFB operates under computer control, in essence, necessitating a runtime interpreter. Since an activation sequence and its association with a basic block or CFG edge is part of a data structure that the interpreter will process, including control flow transfers, it is non-problematic to associate electrode activation sequences with control flow edges, despite the fact that this is not possible in a more traditional setting.

As a potential optimization, consider a CFG edge (b_i, b_j), critical edge splitting can be avoided if b_j is the sole successor of b_i, and/or b_i is the sole predecessor of b_j. In either case, it may be beneficial to move the droplet transport operations associated with $\Sigma_{(b_i,b_j)}$ into Σ_{b_i} or Σ_{b_j} as appropriate: these transport operations could be carried out concurrently with other droplets being routed. To get the best solution, it may be necessary to recompute the routing solutions for b_i and b_j to include the additional droplets. Doing so has the potential to reduce droplet transport latency, although we do not explore this option here.

6.5 Discussion

The preceding subsections intentionally described the stages of the compiler in a manner that was not tied to any specific algorithms for scheduling, placement, or routing. This decision was made because numerous papers have been published already that describe algorithms that could be integrated into our compiler with minimal modification. The objective of this work is to outline the software architecture of the compiler, not to determine which optimization algorithms yield the best results.

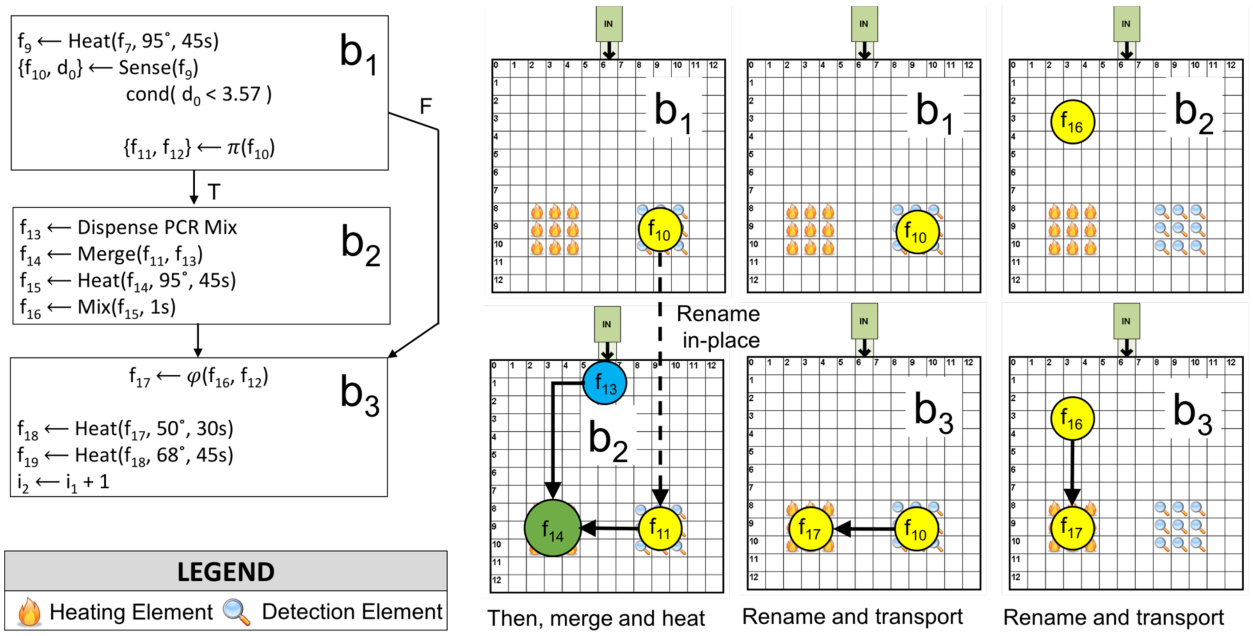

Figure 13. (a) A fragment of the SSI Form CFG taken from Fig. 11. The droplet transport operations inserted for the following CFG edges: (b) (b_1, b_2), (c) (b_1, b_3), and (d) (b_2, b_3).

6.6 Technological Constraints

Unlike a CPU, a DMFB does not contain a fluidic memory hierarchy or any form of off-chip storage. In principle, this may change as liquid handling robots emerge, but for now, we assume that the status quo will persist. Unlike a register allocator for a traditional compiler, this constraint means that a DMFB compiler cannot spill excess fluid off-chip. DMFB compilation may fail, especially at the scheduling stage [58, 59, 66], if demand for spatial resources exceeds on-chip capacity.

These constraints have also influenced our design of the BioCoder language. For example, we do not support function calls because we cannot push fluids that are live across the call onto a non-existent stack; or arrays, which provide the abstraction of (near-)infinite storage, whereas realistic DMFBs are limited to tens of drops on-chip. Attempts to address these issues are left open for future work.

7 Simulation Results

7.1 Compiler and Simulator

The BioCoder compiler described in this paper was built as a set of extensions to an open source and publicly available research infrastructure for programmable microfluidics, released by the University of California, Riverside [58, 98]. The core of the infrastructure is a framework for the evaluation of DMFB scheduling, placement, and routing algorithms, built using common data structures and interfaces. The framework includes a cycle-accurate DMFB simulator that can estimate the execution time of a bioassay. The simulator includes a visualizer that produces an image for each cycle of execution (1 cycle = 10ms); images can then be stitched together to produce animated videos of the DMFB under simulation. The current framework release compiles one basic block, specified as a DAG, at a time.

For testing and debugging purposes, the simulator's execution engine can produce a trace that lists the CFG nodes that were executed, in-order, along with the evaluation of each conditional statement. This way, if something goes wrong, the user can determine precisely which conditions led to the problem, which can help with error diagnosis. For example, an incorrect result could occur because of a faulty sensor or a contaminated sample, among other possible causes.

We modified the compiler and simulator extensively to produce the results reported in this paper. First, we built a front-end parser for the BioCoder Language, which produces an abstract syntax tree (AST). We then convert the AST to a CFG, representing each basic block as DAG. We then compile the CFG, as described in Section 6 of this paper, yielding electrode activation sequences for each basic block and CFG edge, which are then passed to the simulator. We extended the simulator's execution engine to produce videos for CFGs, in addition to DAGs. The simulator also reports total bioassay execution time.

The simulator's execution engine generates pseudo-random numbers as a proxy for sensor readings: the maximum and minimum value for each sensor is specified as part of the configuration file; we do not assume any statistical distribution of sensor values beyond whatever inherent bias may exist in the pseudo-random number generator.

7.2 Methodology

We compiled each benchmark using the heuristics outlined in a previous paper by Grissom and Brisk [66]. These heuristics tend to be fast and greedy, with the added benefit that placement and routing are guaranteed to be successful as long as a legal schedule can be found. As noted earlier, the compiler has been designed to be compatible with any scheduler, placement, and routing heuristic; prior work has already engaged in extensive comparisons among heuristics for these problems [50, 57-85].

For each benchmark, we simulated a 15×19 DMFB with four integrated sensors, two integrated heaters, and fourteen I/O reservoirs placed on the perimeter of the chip (five on the left side, five on top, four on the right side). These resources are sufficient to execute all of the benchmark assays, described next. We assume a 10ms cycle time [70], the time required to transport a droplet from one electrode to its neighbor. We assume that droplets can move horizontally/vertically, but not diagonally.

7.3 Benchmarks and Results

Benchmarking microfluidic technologies is challenging because public repositories of readily usable, relevant, and executable benchmarks simply do not exist. Researchers in the life sciences typically summarize assays in the Materials and Methods sections of peer-reviewed literature, but often do so at a coarse granularity, under the assumption that experts understand and can infer the details, which are learned through training and apprenticeship, further complicating matters.

We performed a thorough literature survey on the design and use of digital microfluidics in the life sciences. In addition to the hierarchical opiate detection immunoassay [51-53] shown in Fig. 5, we obtained two feedback-driven assays that were shown by others to be compatible with cyber-physical DMFB technology. The first is a probabilistic implementation of the polymerase chain reaction (PCR) which terminates early when it senses that a droplet has insufficient initial product to amplify [99]; the second, which was shown in Figs. 10 and 11, is a PCR implementation that performs droplet replenishment to periodically replace fluid volumes lost due to evaporation [89]. We also report results for three assays which do not feature online feedback: image probe synthesis, neurotransmitter sensing, and (vanilla) PCR [3]. The simulator produced animated videos of the simulated execution of the seven bioassays. Source code of the assays and animated videos are online as ACM Digital Library supplementary material.

Table 1 summarizes these assays along with the execution times reported by the simulator. Execution times for a given assay will vary, depending on sensor readings (in real-life), which we simulate with random number generation. For example, we see almost a 4x difference in runtime for the hierarchical opiate detection immunoassay, depending on whether or not the result is positive or negative; similarly, we observe a faster runtime when probabilistic PCR terminates early, failing to amplify DNA. We report only one data point for PCR with droplet replenishment; in practice, the runtime depends on environmental conditions that affect the evaporation rate: faster evaporation increases the replenishment rate, which adds overhead to the assay. The execution times of the three assays that do not feature control flow are presented as-is.

PCR, Probabilistic PCR, and PCR w/droplet replenishment assays come from three distinct sources. The setup and runtimes reported here are based on data from the papers that introduced them. PCR and Probabalistic PCR take ~11 minutes to execute, while PCR w/droplet replenishment takes around 40 minutes. These differences are due to variations in experiment design: the number of thermocycles and the time spent at each temperature.

These results validate the correctness of the compiler and simulator. Our objective was never to evaluate the performance of the various optimization algorithms that are presently built into the compiler. We also do not have any evidence that these assays were particularly challenging to compile, or that they stressed the optimization algorithms; they were chosen because they have been validated in wet laboratory settings by others.

Table 1. Benchmark assays and simulated execution times.
(P/N: positive/negative Opiate detection outcome)
(F/EE: full/early-exit Probabilistic PCR outcome)

Benchmark	Source	Simulated Exec. Time
Opiate detection immunoassay	[51-53]	P 405m 30s N 101m 48s
Probabilistic PCR	[99]	F 11m 19s EE 7m 21s
PCR w/droplet replenishment	[89]	40m 44s
Image probe synthesis	[3]	8m 45s
Neurotransmitter sensing	[3]	05m 59s
PCR	[3]	11m 43s

Providing a programming language and compiler for DMFB technology will lower barriers to entry for practitioners, and will ease the process of designing and validating assays of increasingly complexity, which one day may present a greater challenge to the compiler and its optimization algorithms.

8 Related Work

Many of the languages that have been developed for programmable biochemistry address the $28 billion reproducibility crisis in the life sciences [100]. A widely accepted standardized language with unambiguous syntax could evolve into a de facto standard for dissemination of laboratory procedures. We are hopeful that (1) domain specific programming languages, including but not limited to BioCoder, will improve accessibility to LoC technology among practitioners and lower barriers to entry; and as a side effect, (2) standardize dissemination of LoC-compatible assays, which aligns with the objectives of BioCoder's original development team [55]. For now, there is a wide, arguably insurmountable, chasm between the public sharing of source code in the domain of computing, and standard practices in the biological sciences.

BioCoder is another entry into a much larger domain, although its unique emphasis on DMFB execution distinguish it from prior efforts. Dissemination of laboratory procedures can complement peer-reviewed scientific manuscripts; however, this approach will not make sense if it becomes an extra writing burden on already-overworked scientists. The advantage of a language like BioCoder is that the unambiguous specification also automates the execution of the laboratory procedure.

8.1 Languages for Laboratory Automation

Aquarium [101] is a language for the specification and composition of laboratory workflows using a standard inventory, combining formal and informal statements with photographs. The researcher devises protocols, which are combined to form processes, which are parallelized and scheduled on the available laboratory equipment. BioCoder could integrate with Aquarium. Aquarium's inventory would need to be expanded to include a programmable LoC such as a DMFB, and BioCoder assays could be included as Aquarium processes. Aquarium could then schedule assays on the DMFB in the lab and instruct the technicians which assay to perform on each device.

Cloud-based laboratory automation allows scientists to remotely execute biological experiments in a robot-run laboratory over the Internet. The experiments are described using DSLs that are tailored to the laboratory.

Two companies operating in this space are Transcriptic and Synthace, whose languages Autoprotocol [102] and Antha [103] are based on Python and Go. In principle, these languages could be extended to encompass programmable LoCs such as DMFBs as laboratory components; however, they still need to interact with another language such as BioCoder to execute the assay.

8.2 Device-Specific Languages for SP-LoCs

Programming languages have been designed for specific SP-LoCs which have specialized instruction set architectures (ISAs). BioStream [104], for example, targeted an SP-LoC based on integrated microvalve technology [2], which coupled a fluidic mixer to a fluidic memory. This SP-LoC was performed assays, which dilute one (or more) samples of fluid down to a user-specific concentration. The BioStream language was expressive, but lacked extensibility; to the best of our knowledge, its specification and compiler were never formally released.

Aquacore is an SP-LoC that features a set of fluidic components connected to a centralized bus, and is programmed in an assembly language called the Aquacore Instruction Set (AIS) [3]. AIS instructions activate and deactivate components, and control fluid transfers. Like BioCoder, AIS is highly tied to the device architecture; however, it provides a much lower level of abstraction, as the programmer needs to know precisely how many components exist in the chip in order to program it. In the context of DMFBs, BioCoder's compiler abstracts these low-level details away. It is feasible to create a new variant of BioCoder that targets AquaCore instead of DMFBs and to write a compiler whose back-end emits AIS code, rather than a DMFB executable.

Several other SP-LoCs have been designed and evaluated [4-6, 8], also based on microvalve technologies, but lack high-level language support or a workable software stack. These devices are effectively programmed at the binary level of abstraction: each valve is turned on/off over time, similar to the electrode activation sequences of a DMFB. These devices are controlled in wet laboratories using LabView. Much like AquaCore, it is more than feasible to create a new version of BioCoder and its compiler that is specialized to these targets.

8.3 BioCoder Interpreter

This paper builds on an earlier version of the BioCoder software infrastructure, which was interpreted, rather than compiled [48, 49]. A rudimentary front-end compiler compiled the BioCoder specification to a CFG, and then passed the CFG to the interpreter. As depicted in Fig. 14, the interpreter just-in-time (JIT)-compiles each basic block on-the-fly prior and then executes it immediately; resolving the condition at the end of the basic block determines the next one to execute.

The foremost limitation of this approach is that the assay is paused during each call to the JIT compiler, which forces it to execute low-overhead scheduling, placement, and routing heuristics, which produce relatively poor solution qualities. Moving compilation offline enables the usage of more aggressive and longer running optimization methods that yield higher solution quality. Examples from prior literature include metaheuristics such as simulated annealing [63, 80], evolutionary algorithms [57, 58, 81, 82], tabu search [83, 84] integer linear programming [50, 58, 62, 64], and SAT/SMT formulations [74-76, 85], which have already been compared to greedy heuristics. Our objective is not to re-evaluate these methods in the context of our compiler, but to demonstrate the feasibility of static compilation for cyber-physical DMFBs.

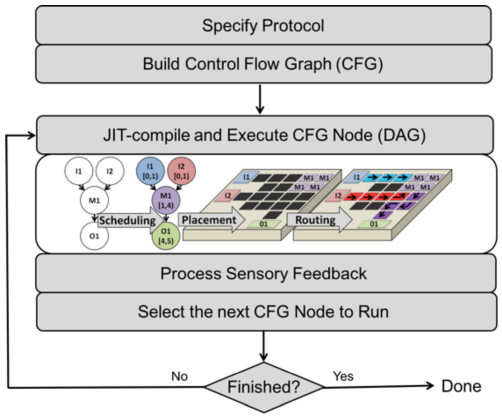

Figure 14. Dynamic interpretation scheme with an integrated JIT-compiler targeting a cyber-physical DMFB [49, Fig. 5].

8.4 Optimization for Cyber-physical DMFBs

Software for cyber-physical DMFBs has emphasized online error detection and recovery for scheduled, placed, and routed DAGs [27, 29-39]. Prior work has shown that soft error detection can be integrated into the high-level assay specification using BioCoder [48]. In contrast, hard errors render portions of the device unusable and require (1) re-execution of any program slices that produced droplets that were lost due to the error; and (2) dynamic re-compilation of the new slices, and any remaining unexecuted portion of the assay, to avoid the unusable portions of the device [27, 32, 33, 35, 36]. In the future, these techniques could be generalized from DAGs (basic blocks) to CFGs and integrated into the BioCoder runtime execution engine.

9 Conclusion and Future Work

This paper introduced an updated version of the BioCoder language, specialized for DMFBs, and established the feasibility of compiling programs that feature control flow operations. The next step in this research is to connect the compiler and execution engine to a real-world device, a conceptually straightforward, but technically challenging, engineering task. As noted in Section 8.4, a real-world platform will likely require online error detection and recovery schemes that are cognizant of the CFG, rather than the basic block, execution model.

Another important task is to promote the technology among life science practitioners. This may lead to refinements to the BioCoder syntax, or perhaps a replacement language. More importantly, we hope that expanding the user base will lead to new applications which are larger and more complex in scope than the assays that were used for evaluation here; they could drive the development of more extensive back-end optimization.

Future work will explore language design issues, such as a fluidic type system and supporting functions, arrays, and threads; it is unclear how to support these abstractions, given the relatively small fluid capacity of a DMFB compared modern computing systems. Longer-term, we hope to integrate BioCoder into a larger cloud laboratory system, which can couple a DMFB (or other fluidic LoC) to a refrigerated storage system, managed by liquid handling robots, i.e., an off-chip fluid storage hierarchy.

Acknowledgments

This work was partially supported by NSF Awards #1351115, #1536026, #1545097, #1640757, and #1740052.

References

[1] M. Pollack, A. Shenderov, and R. Fair. 2002. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab-on-a-Chip* 2, 2 (May, 2002), 96-101. DOI: http://dx.doi.org/10.1039/B110474H

[2] J. P. Urbanski, W. Thies, C. Rhodes, S. Amarasinghe, and T. Thorsen. 2006. Digital microfluidics using soft lithography. *Lab-on-a-Chip* 6, 1, (Jan. 2006), 96–104. DOI: http://dx.doi.org/10.1039/B510127A

[3] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson. 2007. Aquacore: A programmable architecture for microfluidics. In *Proceedings of the 34th International Symposium on Computer Architecture* (ISCA'07). 254-265. DOI: http://dx.doi.org/10.1145/1250662.1250694

[4] E. C. Jensen, B. P. Bhat, and R. A. Mathies. 2010. A digital microfluidic platform for the automation of quantitative biomolecular assays. *Lab-on-a-Chip* 10, 6 (Mar. 2010), 685–691. DOI: http://dx.doi.org/10.1039/b920124f

[5] L. M. Fidalgo and S. J. Maerkl. 2011. A software-programmable microfluidic device for automated biology. *Lab-on-a-Chip* 11, 9 (May, 2011), 1612–1619, DOI: http://dx.doi.org/10.1039/C0LC00537A

[6] K. Leung, H. Zahn, T. Leaver, K. M. Konwar, N. W. Hanson, A. P. Page, C-C. Lo, P. S. Chain, S. J. Hallam, and C. L. Hansen. 2012. A programmable droplet-based microfluidic device applied to multiparameter analysis of single microbes and microbial communities. *Proceedings of the National Academies of Sciences of the United States of America* 109, 20 (May 2012). 7665-7670. DOI: http://dx.doi.org/10.1073/pnas.1106752109

[7] A. M. Amin, R. Thakur, S. Madren, H. S. Chuang, M. Thottethodi, T. N. Vijaykumar, S. T. Wereley, and S. C. Jacobson. 2013. Software-programmable continuous-flow multi-purpose lab-on-a-chip. *Microfluidics and Nanofluidics* 15, 5 (Nov. 2013), 657-659. DOI: http://dx.doi.org/10.1007/s10404-013-1180-2

[8] G. Linshiz, E. Jensen, N. Stawski, C. Bi, N. Elsbree, H. Jiao, J. Kim, R. Mathies, J. D. Keasling, and N. J. Hillson. 2016. End-to-end automated microfluidic platform for synthetic biology: from design to functional analysis. *Journal of Biological Engineering* 10 (Feb. 2016), Article #3. DOI: https://dx.doi.org/10.1186/s13036-016-0024-5

[9] H. Ren, R. B. Fair, and M. G. Pollack. 2004. Automated on-chip droplet dispensing with volume control by eletrowetting actuation and capacitance metering. *Sensors and Actuators B: Chemical* 98, 2-3 (Mar. 2004). 319-327. DOI: http://dx.doi.org/10.1016/j.snb.2003.09.030

[10] V. Srinivasan, V. K. Pamula, and R. B. Fair. Droplet-based microfluidic lab-on-a-chip for glucose detection. *Analytica Chimica Acta* 507, 1 (Apr. 2004) 145-150. DOI: http://dx.doi.org/10.1016/j.aca.2003.12.030

[11] J. Gong and C-J. Kim. 2008. All-electronic droplet generation on-chip with real-time feedback control for EWOD digital microfluidics. *Lab-on-a-Chip* 8, 6 (June 2008). 898-906. DOI: http://dx.doi.org/10.1039/b717417a

[12] S. C. C. Shih, R. Fobel, P. Kumar, and A. R. Wheeler. 2011. A feedback control system for high-fidelity digital microfluidics. *Lab-on-a-Chip* 11, 3 (Feb. 2011). 535-540. DOI: http://dx.doi.org/10.1039/c0lc00223b

[13] S. Sadeghi, H. Ding, G. J. Shah, S. Chen, P. Y. Keng, C-J. Kim, and R. M. van Dam. 2012. On chip droplet characterization: a practical, high-sensitivity measurement of droplet impedance in digital microfluidics. *Analytical Chemistry* 84, 4 (Jan. 2012). 1915-1923. DOI: http://dx.doi.org/10.1021/ac202715f

[14] M. J. Schertzer, R. Ben Mrad, and P. E. Sullivan. 2012. Automated detection of particle concentration and chemical reactions in EWOD devices. *Sensors and Actuators B: Chemical* 164, 1 (Mar. 2012). 1-6. DOI: http://dx.doi.org/10.1016/j.snb.2012.01.027

[15] M. Murran and H. Najjaran. 2012. Capacitance-based droplet position estimator for digital microfluidic devices. *Lab-on-a-Chip* 12, 11 (Mar. 2012). 2053-2059. DOI: http://dx.doi.org/10.1039/C2LC21241B

[16] L. Luan, M. W. Royal, R. Evans, R. B. Fair, and N. M. Jokerst. 2012. Chip scale optical microresonator sensors integrated with embedded thin film photodetectors on electrowetting digital microfluidics platforms. *IEEE Sensors Journal* 12, 6 (June 2012). 1794-1800. DOI: http://dx.doi.org/10.1109/JSEN.2011.2179027

[17] T. Lederer, S. Clara, B. Jakoby, and W. Hilber. 2012. Integration of impedance spectroscopy sensors in a digital microfluidic platform. *Microsystem Technologies* 18, 7-8 (Aug. 2012). 1163-1180. DOI: http://dx.doi.org/10.1007/s00542-012-1464-6

[18] B. Bhattacharjee and H. Najjaran. 2012. Droplet sensing by measuring the capacitance between coplanar electrodes in a digital microfluidic system. *Lab-on-a-Chip* 12, 21 (Nov. 2012). 4416-4423. DOI: http://dx.doi.org/10.1039/c2lc40647k

[19] J. Gao, X. Liu, T. Chen, P. I. Mak, Y. Du, M. I. Vai, B. Lin, and R. P. Martins. 2013. An intelligent digital microfluidic system with fuzzy-enhanced feedback for multi-droplet manipulation. *Lab-on-a-Chip* 13, 3 (Feb. 2013). 443-451. DOI: http://dx.doi.org/10.1039/c2lc41156c

[20] S. C. Shih, I. Barbulovic-Nad, X. Yang, R. Fobel, and A. R. Wheeler. 2013. Digital microfluidics with impedance sensing for integrated cell culture and analysis. *Biosensors and Bioelectronics* 42, (Apr. 2013). 314-320. DOI: http://dx.doi.org/10.1016/j.bios.2012.10.035

[21] M. W. Royal, N. M. Jokerst, and R. B. Fair. 2013. Droplet-based sensing: optical microresonator sensors embedded in digital electrowetting microfluidics systems. *IEEE Sensors Journal* 13, 12 (Dec. 2013). 4733-4742. DOI: http://dx.doi.org/10.1109/JSEN.2013.2273828

[22] Y. Li, H. Li, and R. Baker. 2014. Volume and concentration identification by using an electrowetting on dielectric device. In *Proceedings of the IEEE Dallas Circuits and Systems Conference (DCAS'14)*. 1–4. DOI: http://dx.doi.org/10.1109/DCAS.2014.6965350

[23] Y. Li, H. Li, and R. J. Baker. 2015. A low-cost and high-resolution droplet position detector for an intelligent electrowetting on dielectric device. *Journal of Laboratory Automation* 20, 6 (Jan. 2015). 663-669. DOI: http://dx.doi.org/10.1177/2211068214566940

[24] Y. Shin and J. Lee. 2010. Machine vision for digital microfluidics. *Review of Scientific Instruments* 81, 1 (Jan. 2010). 014302. DOI: http://dx.doi.org/10.1063/1.3274673

[25] A. S. Basu. 2013. Droplet morphometry and velocimetry (dmv): a video processing software for time-resolved, label-free tracking of droplet parameters," *Lab-on-a-Chip* 13, 10 (May 2013). 1892-1901. DOI: http://dx.doi.org/10.1039/c3lc50074h

[26] R. Fobel, C. Fobel, and A. R. Wheeler. 2013. Dropbot: An open source digital microfluidic control system with precise control of electrostatic driving force and instantaneous drop velocity measurement. *Applied Physics Letters* 102, 19 (May 2013). 193513. DOI: http://dx.doi.org/10.1063/1.4807118

[27] K. Hu, B. Hsu, A. Madison, K. Chakrabarty, and R. B. Fair. 2013. Fault detection, real-time error recovery, and experimental demonstration for digital microfluidic biochips. In *Proceedings of Design Automation and Test in Europe* (DATE'13). 559-564. DOI: http://dx.doi.org/10.7873/DATE.2013.124

[28] P. Q. N. Vo, M. C. Husser, F. Ahmadi, H. Sinha, and S. C. C. Shih. 2017. Image-based feedback and analysis system for digital microfluidics. *Lab-on-a-Chip* 17, 20, (Sep. 2017), 3437-3446. DOI: http://dx.doi.org/10.1039/c7lc00826k

[29] Z. Li, K. Y-T. Lai, J. McCrone, P-H. Yu, K. Chakrabarty, M. Pajic, T-Y. Ho, and C-Y. Lee. 2017. Efficient and adaptive error recovery in a micro-electrode-dot-array digital microfluidic biochip. *IEEE TCAD* (July 2017). Preprint. DOI: http://dx.doi.org/10.1109/TCAD.2017.2729347

[30] Y. Zhao, T. Xu, and K. Chakrabarty. 2010. Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-a-chip. *ACM JETC* 6, 3 (Aug. 2010). Article #11. DOI: http://dx.doi.org/10.1145/1777401.1777404

[31] M. Alistar, P. Pop, and J. Madsen. 2012. Online synthesis for error recovery in digital microfluidic biochips with operation variability. In *Proceedings on the 2012 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS* (DTIP'12).

[32] Y. Luo, K. Chakrabarty, and T-Y. Ho. 2013. Error recovery in cyberphysical digital microfluidic biochips. *IEEE TCAD* 32, 1 (Jan. 2013). 59-72. DOI: http://dx.doi.org/10.1109/TCAD.2012.2211104

[33] Y. Luo, K. Chakrabarty, and T-Y. Ho. 2013. Real-time error recovery in cyberphysical digital-microfluidic biochips using a compact dictionary. *IEEE TCAD* 32, 12 (Dec. 2013). 1839-1852. DOI: http://dx.doi.org/10.1109/TCAD.2013.2277980

[34] Y-L. Hsieh, T-Y. Ho, and Krishnendu Chakrabarty. 2014. Biochip synthesis and dynamic error recovery for sample preparation using digital microfluidics. *IEEE TCAD* 33, 2 (Feb. 2014). 183-196. DOI: http://dx.doi.org/10.1109/TCAD.2013.2284010

[35] M. Ibrahim and K. Chakrabarty. 2015. Error recovery in digital microfluidics for personalized medicine. In *Proceedings of Design Automation and Test in Europe* (DATE'15). 247-252. DOI: http://dx.doi.org/10.7873/DATE.2015.1126

[36] C. Jaress, P. Brisk, and D. T. Grissom. 2015. Rapid online fault recovery for cyber-physical digital microfluidic biochips. In *Proceedings of the IEEE VLSI Test Symposium* (VTS'15). 1-6. DOI: http://dx.doi.org/10.1109/VTS.2015.7116246

[37] M. Alistar and P. Pop. 2015. Synthesis of biochemical applications on digital microfluidic biochips with operation execution time variability. *Integration: The VLSI Journal* 51, (Sep. 2015) 158-168. DOI: http://dx.doi.org/10.1016/j.vlsi.2015.02.004

[38] S. Poddar, S. Ghoshal, K. Chakrabarty, and B. B. Bhattacharya. 2016. Error-correcting sample preparation with cyberphysical digital microfluidic lab-on-chip. *ACM TODAES* 22, 1 (July 2016), Article #2. DOI: http://dx.doi.org/10.1145/2898999

[39] M. Ibrahim, K. Chakrabarty, and K. Scott. 2017. Synthesis of cyberphysical digital-microfluidic biochips for real-time quantitative analysis. *IEEE TCAD* 36, 5 (May, 2017). 733-746. DOI: http://dx.doi.org/10.1109/TCAD.2016.2600626

[40] H. Moon, S. K. Cho, R. L. Garrell, and C-J. Kim. 2002. Low voltage electrowetting-on-dielectric. *Journal of Applied Physics* 92, 7 (Sep. 2002). DOI: http://dx.doi.org/10.1063/1.1504171

[41] J. Gong and C-J. Kim. 2008. Direct-referencing two-dimensional array digital microfluidics using multilayer printed circuit board. *Journal of Microelectromechanical Systems* 17, 2 (Apr. 2008). 257-264. DOI: http://dx.doi.org/10.1109/JMEMS.2007.912698

[42] G. Wang, D. Teng, and S-K. Fan. 2011. Digital microfluidic operations on micro-electrode dot array architecture. *IET Nanobiotechnology* 5, 4 (Dec. 2011). 152-160. DOI: http://dx.doi.org/10.1049/iet-nbt.2011.0018

[43] J. H. Noh, J. Noh, E. Kreit, J. Heikenfeld, and P. D. Rack. 2012. Toward active-matrix lab-on-a-chip: programmable electrofluidic control enabled by arrayed oxide thin film transistors. *Lab-on-a-Chip* 2, 2 (Jan. 2012). 353–360. DOI: http://dx.doi.org/10.1039/c1lc20851a

[44] B. 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-on-a-Chip* 12, 18 (Sep. 2012). 3305-3313. DOI: http://dx.doi.org/10.1039/c2lc40273d

[45] G. Wang, D. Teng, Y-T. Lai, Y-W. Lu, Y. Ho, and C-Y. Lee. 2014. Field-programmable lab-on-a-chip based on microelectrode dot array architecture. *IET Nanobiotechnology* 8, 3 (Sep. 2014). 163-171. DOI: http://dx.doi.org/10.1049/iet-nbt.2012.0043

[46] A. Banerjee, J. H. Noh, Y. Liu, P. D. Rack, and I. Papautsky. 2015. Programmable electrowetting with channels and droplets. *Micromachines* 6, 2 (Jan. 2015). 172-185. DOI: http://dx.doi.org/10.3390/mi6020172

[47] S. Kalsi, M. Valiadi, M. N. Tsaloglou, L. Parry-Jones, A. Jacobs, R. Watson, C. Turner, R. Amos, B. Hadwen, J. Buse, C. Brown, M. Sutton, and H. Morgan. 2015. apid and sensitive detection of antibiotic resistance on a programmable digital microfluidic platform. *Lab-on-a-Chip* 5, 14 (Jul. 2015). 3065-3075. DOI: http://dx.doi.org/10.1039/c5lc00462d

[48] D. Grissom, C. Curtis, and P. Brisk. 2014. Interpreting assays with control flow on digital microfluidic biochips. *ACM JETC* 10, 3 (Apr. 2014). Article #24. DOI: http://dx.doi.org/10.1145/2567669

[49] C. Curtis and P. 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 (Dec. 2015). 110-116. DOI: http://dx.doi.org/10.1016/j.mee.2015.10.007

[50] J. Ding, K. Chakrabarty, and R. B. Fair. 2001. Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *IEEE TCAD* 20, 12 (Dec. 2001). 1463-1468. DOI: http://dx.doi.org/10.1109/43.969439

[51] R. C. Backer, J. R. Monforte, and A. Poklis. 2005. Evaluation of the DRI oxycodone immunoassay for the detection of oxycodone in urine. *Journal of Analytical Toxicology* 29, 7 (Oct. 2005). 675-677.

[52] C. L. Mao, K. D. Zientek, P. T. Colahan, M. Y. Kuo, C. H. Liu, K. M. Lee, and C. C. Chou. 2006. Development of an enzyme linked immunosorbent assay for fentanyl and applications of fentanyl antibody-coated nanoparticles for sample preparation. *Journal of Pharmaceutical and Biomedical Analysis* 41, 4 (June 2006). 1332-1341. DOI: http://dx.doi.org/10.1016/j.jpba.2006.03.009

[53] Y. Jiang, X. Huang, K. Hu, W. Yu, X. Yang, and L. Lv. 2011. Production and characterization of monoclonal antibodies against small hapten-ciprofloxacin. *African Journal of Biotechnology* 10, 65 (2011). 14342-14347. DOI: http://dx.doi.org/10.5897/AJB11.1546

[54] E. Miller, A. H. C. Ng, U. Uddayasankar, and A. Wheeler. 2011. A digital microfluidic approach to heterogeneous immunoassays. Analytical and Bioanalytical Chemistry 339, 1 (Jan. 2011). 337–345. DOI: http://dx.doi.org/10.1007/s00216-010-4368-2

[55] V. Ananthanarayanan and W. Thies. 2010. Biocoder: A programming language for standardizing and automating biology protocols. *Journal of Biological Engineering* 4 (Nov. 2010). Article #13. DOI: http://dx.doi.org/10.1186/1754-1611-4-13

[56] D. Grissom, C. Curtis, S. Windh, C. Phung, N. Kumar, Z. Zimmerman, O. Kenneth, J. McDaniel, N. Liao, and P. Brisk. 2015. An open-source compiler and pcb synthesis tool for digital microfluidic biochips. *Integration: the VLSI Journal* 51 (Sep. 2015). 169-193. DOI: http://dx.doi.org/10.1016/j.vlsi.2015.01.004

[57] A. J. Ricketts, K. M. Irick, N. Vijaykrishnan, and M. J. Irwin. 2006. Priority scheduling in digital microfluidics-based biochips. In *Proceedings of Design, Automation and Test in Europe* (DATE'06). 329-334. DOI: http://dx.doi.org/10.1109/DATE.2006.244178

[58] F. Su and K. Chakrabarty. 2008. High-level synthesis of digital microfluidic biochips. *ACM JETC* 3, 4 (Jan. 2008). Article #1. DOI: http://dx.doi.org/10.1145/1324177.1324178

[59] D. Grissom and P. Brisk. 2012. Path scheduling on digital microfluidic biochips. In *Proceedings of the Design Automation Conference* (DAC'12). 26-35. DOI: http://dx.doi.org/10.1145/2228360.2228367

[60] K. O'Neal, D. Grissom, and P. Brisk. 2012. Force-directed list scheduling for digital microfluidic biochips. in *Proceedings of the 20th IEEE/IFIP International Conference on VLSI and System-on-Chip* (VLSI-SoC'12). 7-11. DOI: http://dx.doi.org/10.1109/VLSI-SoC.2012.7332068

[61] C. Liu, K. Liu, and J. Huang. 2013. Latency-optimization synthesis with module selection for digital microfluidic biochips. In *Proceedings of the IEEE International SOC Conference* (SOCC'13). 159-164. DOI: http://dx.doi.org/10.1109/SOCC.2013.6749681

[62] A. Yadav, T. A. Dinh, D. Kitagawa, and S. Yamashita. 2016. ILP-based synthesis for sample preparation applications on digital microfluidic biochips. In *Proceedings of the 29th International Conference on VLSI Design* (VLSID'16). DOI: http://dx.doi.org/10.1109/VLSID.2016.41

[63] F. Su and K. Chakrabarty. 2006. Module placement for fault-tolerant microfluidics-based biochips. *ACM TODAES* 11, 3 (July 2006). 682-710. DOI: http://dx.doi.org/10.1145/1142980.1142987

[64] C. Liao and S. Hu. 2011. Multiscale variation-aware techniques for high-performance digital microfluidic lab-on-a-chip component placement. *IEEE Transactions on Nanobioscience* 10, 1 (Mar. 2011). 51-58. DOI: http://dx.doi.org/10.1109/TNB.2011.2129596

[65] Y. Chen, C. Hsu, L. Tsai, T. Huang, and T. Ho. 2013. A reliability-oriented placement algorithm for reconfigurable digital microfluidic biochips using 3-d deferred decision making technique. *IEEE TCAD* 32, 8 (Aug. 2013). 1151-1162. DOI: http://dx.doi.org/10.1109/TCAD.2013.2249558

[66] D. Grissom and P. Brisk. 2014. Fast online synthesis of digital microfluidic biochips. *IEEE TCAD* 33, 3 (Mar. 2014). 356-369. DOI: http://dx.doi.org/10.1109/TCAD.2013.2290582

[67] F. Su, W. L. Hwang, and K. Chakrabarty. 2006. Droplet routing in the synthesis of digital microfluidic biochips. in *Proceedings of Design, Automation and Test in Europe* (DATE'06). 323-328. DOI: http://dx.doi.org/10.1109/DATE.2006.244177

[68] K. Bohringer. 2006. Modeling and controlling parallel tasks in droplet-based microfluidic systems. *IEEE TCAD* 25, 2 (Feb. 2006). 334-344. DOI: http://dx.doi.org/10.1109/TCAD.2005.855958

[69] M. Cho and D. Z. Pan. 2008. A high-performance droplet routing algorithm for digital microfluidic biochips. *IEEE TCAD* 27, 10 (Oct. 2008). 1714-1724. DOI: http://dx.doi.org/10.1109/TCAD.2008.2003282

[70] P. Yuh, C. Yang, and Y. Chang. 2008. BioRoute: A network-flowbased routing algorithm for the synthesis of digital microfluidic biochips. *IEEE TCAD* 27, 11 (Nov. 2008). 1928-1941. DOI: http://dx.doi.org/10.1109/TCAD.2008.2006140

[71] T. Huang and T. Ho. 2009. A fast routability- and performancedriven droplet routing algorithm for digital microfluidic biochips. In *Proceedings of the 27th International Conference on Computer Design* (ICCD'09). 445-450. DOI: http://dx.doi.org/10.1109/ICCD.2009.5413119

[72] P. Roy, H. Rahaman, and P. Dasgupta. 2010. A novel droplet routing algorithm for digital microfluidic biochips. In *Proceedings of the 20th ACM Great Lakes Symposium on VLSI* (GLSVLSI'09). 441-446. DOI: http://dx.doi.org/10.1145/1785481.1785583

[73] P. Roy, H. Rahaman, and P. Dasgupta. 2012. Two-level clustering-based techniques for intelligent droplet routing in digital microfluidic biochips. *Integration: The VLSI Journal* 45, 3 (June, 2012). 316-330. DOI: http://dx.doi.org/10.1016/j.vlsi.2011.11.006

[74] O. Keszocze, R. Wille, and R. Drechsler. 2014. Exact routing for digital microfluidic biochips with temporary blockages. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (ICCAD'14). 405-410. DOI: http://dx.doi.org/10.1109/ICCAD.2014.7001383

[75] O. Keszocze, R. Wille, K. Chakrabarty, and R. 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'15). 874-881. DOI: http://dx.doi.org/10.1109/ICCAD.2015.7372663

[76] O. Keszocze, Z. Li, A. Grimmer, R. Wille, K. Chakrabarty, and R. Drechsler. 2017. Exact routing for micro-electrode-dot-array digital microfluidic biochips. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference* (ASP-DAC'17). DOI: http://dx.doi.org/10.1109/ASPDAC.2017.7858407

[77] T. Huang, C. Lin, and T. Ho. 2010. A contamination aware droplet routing algorithm for the synthesis of digital microfluidic biochips. *IEEE TCAD* 29, 11 (Nov. 2010). 1682-1695. DOI: http://dx.doi.org/10.1109/TCAD.2010.2062770

[78] Y. Zhao and K. Chakrabarty. 2012. Cross-contamination avoidance for droplet routing in digital microfluidic biochips. *IEEE TCAD* 31, 6 (June 2012). 817-830. http://dx.doi.org/10.1109/TCAD.2012.2183369

[79] H. Yao, Q. Wang, Y. Shen, T. Ho, and Y. Cai. 2016. Integrated functional and washing routing optimization for cross-contamination removal in digital microfluidic biochips. *IEEE TCAD* 35, 8 (Aug. 2016). 1283-1296. DOI: http://dx.doi.org/10.1109/TCAD.2015.2504397

[80] P. Yuh, C. Yang, and Y. Chang. 2007. Placement of defect-tolerant digital microfluidic biochips using the t-tree formulation. *ACM JETC* 3, 3 (Nov. 2007). Article #13. DOI: http://dx.doi.org/10.1145/1295231.1295234

[81] T. Xu and K. Chakrabarty. 2008. Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips. *ACM JETC* 4, 3 (Aug. 2008). Article #11. DOI: http://dx.doi.org/10.1145/1389089.1389091

[82] T. Xu, K. Chakrabarty, and F. Su. Defect-aware high-level synthesis and module placement for microfluidic biochips. *IEEE TBioCAS* 2, 1 (Mar. 2008). 50-62. DOI: http://dx.doi.org/10.1109/TBCAS.2008.918283

[83] E. Maftei, P. Pop, and J. Madsen. 2010. Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices. *Design Automation for Embedded Systems* 14, 3 (Sep. 2010). 287-307. DOI: http://dx.doi.org/10.1007/s10617-010-9059-x

[84] E. Maftei, P. Pop, and J. Madsen. 2013. Module-based synthesis of digital microfluidic biochips with droplet-aware operation execution. *ACM JETC* 9, 1 (Feb. 2013). Article #2. DOI: http://dx.doi.org/10.1145/2422094.2422096

[85] R. Wille, O. Keszocze, R. Drechsler, T. Boehnisch, A. Kroker. 2015. Scalable one-pass synthesis for digital microfluidic biochips. *IEEE Design & Test* 32, 6 (Dec. 2015). 41-50. DOI: http://dx.doi.org/10.1109/MDAT.2015.2455344

[86] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. 1991. Efficiently computing static single assignment form and the control dependence graph. *ACM TOPLAS* 13, 4 (Oct. 1991). 451-490. DOI: http://dx.doi.org/10.1145/115372.115320

[87] J-D. Choi, R. Cytron, J. Ferrante. 1991. Automatic construction of sparse data flow evaluation graphs. In *Proceedings of the International Conference on Principles of Programming Languages* (POPL'91). 55-66. DOI: http://dx.doi.org/10.1145/99583.99594

[88] P. Briggs, K. D. Cooper, T. J. Harvey, and L. T. Simpson. 1998. Practical improvements to the construction and destruction of static single assignment form. *Software: Practice & Experience* 28, 8 (July 1998). 859-881. DOI: http://dx.doi.org/http://dx.doi.org/10.1002/(SICI)1097-024X(19980710)28:8<859::AID-SPE188>3.0.CO;2-8

[89] M. Jebrail, R. Renzi, A. Sinha, J. Van De Vreugde, C. Gondhalekar, C. Ambriz, R. Meagher, and S. Branda. 2015. A solvent replenishment solution for managing evaporation of biochemical reactions in air-matrix digital microfluidics devices. Lab-on-a-Chip 15, 1 (Jan. 2015). 151-158. DOI: http://dx.doi.org/10.1039/c4lc00703d

[90] C. S. Ananian. 1999. The Static Single Information Form. M.S. Thesis. Massachusetts Institute of Technology, Cambridge, MA, USA.

[91] J. Singer. 2005. Static Program Analysis based on Virtual Register Renaming. Ph.D. Thesis. University of Cambridge, UK.

[92] B. Boissinot, P. Brisk, A. Darte, and F. Rastello. 2012. SSI properties revisited. *ACM TECS* 11S, 1 (June 2012). Article #21. DOI: http://dx.doi.org/10.1145/2180887.2180898

[93] K. D. Cooper and L. Torczon. 2004. Engineering a Compiler. Morgan Kaufmann 2004, ISBN 1-55860-699-8

[94] P. Paik, V. K. Pamula, and R. B. Fair. 2003. Rapid droplet mixers for digital microfluidic systems. *Lab-on-a-Chip* 3, 4 (Sep. 2003). 253–259. DOI: http://dx.doi.org/10.1039/b307628h

[95] K. Bazargan, R. Kastner, and M. Sarrafzadeh. 2000. Fast template placement for reconfigurable computing systems. *IEEE Design & Test of Computers* 17, 1 (Jan-Mar.2000). 68-83. DOI: http://dx.doi.org/10.1109/54.825678

[96] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. 1981. Register allocation via coloring. *Computer Languages* 6, 1 (Jan. 1981). 47-57. DOI: http://dx.doi.org/10.1016/0096-0551(81)90048-5

[97] P. Briggs, K. D. Cooper, and L. Torczon. 1994. Improvements to graph coloring register allocation. *ACM TOPLAS* 16, 3 (May, 1994). 428-455. DOI: http://dx.doi.org/10.1145/177492.177575

[98] F. M. Q. Pereira and J. Palsberg. 2008. Register allocation by puzzle solving. In *Proceedings of the 29th SIGPLAN Conference on Programming Language Design and Implementation* (PLDI'08). 216-226. DOI: http://dx.doi.org/10.1145/1375581.1375609

[93] V. C. Sreedhar, R. D-C. Ju, D. M. Gillies, and V. Santhanam. 1999. Translating out of static single assignment form. In *Proceedings of the Static Analysis Symposium* (SAS'99). 194-210. DOI: http://dx.doi.org/10.1007/3-540-48294-6_13

[94] Z. Budimlic, K. D. Cooper, T. J. Harvey, K. Kennedy, T. S. Oberg, and S. W. Reeves. 2002. Fast copy coalescing and live-range identification. In *Proceedings of the 26th SIGPLAN Conference on Programming Language Design and Implementation* (PLDI'02). 25-32. DOI: http://dx.doi.org/10.1145/543552.512534

[95] F. Rastello, F. de Ferrière, and C. Guillon. 2004. Optimizing translation out of SSA using renaming constraints. In *Proceedings of the International Symposium on Code Generation and Optimization* (CGO'04). 265-278. DOI: http://dx.doi.org/10.1109/CGO.2004.1281680

[96] B. Boissinot, A. Darte, F. Rastello, B. D. de Dinechin, and C. Guillon. 2009. Revisiting out-of-SSA translation for correctness, code quality and efficiency. In *Proceedings of the International Symposium on Code Generation and Optimization* (CGO'09). 114-125. DOI: http://dx.doi.org/10.1109/CGO.2009.19

[97] F. M. Q. Pereira and J. Palsberg. 2009. SSA elimination after register allocation. In *Proceedings of Compiler Construction* (CC'09). 158-173. DOI: https://dx.doi.org/10.1007/978-3-642-00722-4_12

[98] UC Riverside Digital Microfluidic Biochip Static Synthesis Simulator. URL: http://microfluidics.cs.ucr.edu/dmfb_static_simulator/overview.html

[99] Y. Luo, B.B. Bhattacharya, T-Y. Ho, and K. Chakrabarty. Design and optimization of a cyberphysical digital-microfluidic biochip for the polymerase chain reaction. *IEEE TCAD* 34, 1 (Jan. 2015). 29-42. DOI: http://dx.doi.org/10.1109/TCAD.2014.2363396

[100] L. Freedman, I. Cockburn, and T. Simcoe. 2015. The economics of reproducibility in preclinical research. *PLoS Biology* 13, 6 (June 2015). e1002165. DOI: http://dx.doi.org/10.1371/journal.pbio.1002165

[101] K. Eric. Aquarium, your protocols will be assimilated. http://klavinslab.org/aquarium.html Accessed: 2011-11-13.

[102] Autoprotocol: an open standard for life science experimental design and automation. URL: http://autoprotocol.org

[103] Antha. URL: https://docs.antha.com

[104] W. Thies, J. P. Urbanski, T. Thorsen, and S. Amarasinghe. Abstraction layers for scalable microfluidic biocomputing. *Natural Computing* 7, 2 (June 2008). 255-275. DOI: http://dx.doi.org/10.1007/s11047-006-9032-6