Time- and Resource-Constrained Scheduling for Digital Microfluidic Biochips

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

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

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

ABSTRACT

Digital microfluidic biochips (DMFBs) are a class of software-programmable laboratories-on-a-chip capable of automating and miniaturizing biochemical assays. Many assays feature time-sensitive interactions which are not supported by existing programming languages or compilers. This paper presents high-level language annotations to allow programmers to specify assays that feature time-sensitive dependencies between operations. To provide compiler support, this paper also presents a new formulation of a microfluidic scheduling problem for DMFBs that enforces timing constraints. To solve the problem, which is NP-complete, the paper introduces an efficient heuristic, along with an optimal Integer Linear Programming (ILP) formulation. Experiments demonstrate that existing methods for the traditional scheduling problem often do not yield solutions that satisfy time-sensitive constraints, while the proposed methods can do so while also effectively minimizing the total execution time of a scheduled assay.

CCS CONCEPTS

• **Software and its engineering** → **Compilers; Domain specific languages**; • **Hardware** → *Emerging languages and compilers; On-chip resource management*.

KEYWORDS

microfluidics, digital microfluidic biochip (DMFB), time-constrained scheduling, resource-constrained scheduling

ACM Reference Format:

Tyson Loveless, Jason Ott, and Philip Brisk. 2021. Time- and Resource-Constrained Scheduling for Digital Microfluidic Biochips. In *ACM/IEEE 12th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2021) (ICCPs '21)*, May 19–21, 2021, Nashville, TN, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3450267.3450547>

1 INTRODUCTION

Programmable *microfluidic* technologies offer the benefits of miniaturization, automation, and programmability to the life sciences: a programmer specifies an *assay* (i.e., a protocol that measures the biochemical or immunological activity of a target *analyte*) using a

domain-specific programming language, which is then compiled into an executable format [1, 9, 14] or interpreted on-the-fly [6, 20]. The most advanced programmable microfluidic platforms are *cyber-physical*, where integrated sensors [12] and/or computer vision systems [17] provide real-time feedback to a desktop or embedded PC which controls the device. Domain-specific programming languages and their compilers enable programmers to specify assays whose control flow depends on sensory feedback [1, 14, 20].

Many assays are *time-sensitive*, although it may not always be obvious from the description. For example, an assay-writer might indicate a *good time to take a break* as a “pause point” [16], implying that breaks *should not* be taken at other points. Reagents can be *extremely sensitive* to temperature changes, evaporation, and exposure to other reagents, especially at sub-microliter volumes [2], and can lead to inaccuracies in reporting or failure in the worst case. *Existing languages presently lack syntactic constructs that could allow the programmer to specify these constraints, and their compilers lack a mechanism to enforce them.*

Today’s programming languages and compilers support *timed operations* (e.g., mix or heat for s seconds) but **do not** support *timing constraints* on the intermediate fluids that are produced and subsequently consumed during assay execution; for example, it may be necessary that a fluid *generated* at time t must be *consumed* before time $t + \Delta$ to prevent spoilage. Existing compilers aim to minimize assay execution time [13], but do not enforce timing constraints. When available on-chip resources are scant, the compiler may temporarily store a time-sensitive intermediate product beyond its usable lifespan.

To address these concerns, this paper makes the following contributions:

- It introduces syntactical annotations that can be added to any microfluidic programming language to allow specification of timing constraints for fluids; the annotations are added to the BioScript language [14] as a proof-of-concept.
- It reformulates the resource-constrained scheduling problem for compilers targeting *Digital Microfluidic Biochips (DMFBs)* (Fig. 1) – a programmable microfluidic technology – and presents a new heuristic and an optimal *Integer Linear Programming (ILP)* formulation to solve this problem.
- It presents two benchmark suites to evaluate a scheduler’s ability to solve the updated scheduling problem, and compares existing scheduling methods against those presented here, demonstrating their effectiveness

The remainder of this paper is organized as follows: § 2 provides the necessary background on DMFBs and languages and compilers for programmable laboratories-on-a-chip. § 3 describes the motivation for timing constraints, and introduces syntactical annotations

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICCPs '21, May 19–21, 2021, Nashville, TN, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8353-0/21/05...\$15.00

<https://doi.org/10.1145/3450267.3450547>

Figure 1: The electrowetting effect (a); Digital Microfluidic Biochip (DMFB) (b) architecture; (c) droplet transport; and (d) instruction set

to associate them with operations, culminating with a proof-of-concept implementation in BioScript. § 4 specifies the scheduling problem for DMFBs and introduces scheduling constraints; § 5 presents heuristic and optimal algorithms to solve the scheduling problem; §§ 6 and 7 summarizes the benchmarks used in our study and presents an experimental evaluation of our proposed methods; § 8 summarizes related work on scheduling in microfluidics, as well as other contexts where timing constraints are likely to occur; finally, § 9 concludes the paper and outlines potential directions for future research.

2 PRELIMINARIES

This paper focuses on one specific programmable microfluidic technology as a proof of concept: *Electrowetting on Dielectric (EWoD)* [8]; chips based on this principle are widely called *Digital Microfluidic Biochips (DMFBs)* in the literature. Fig. 1a depicts the *electrowetting effect*: applying a potential across a liquid droplet reduces its contact angle with the surface, causing the droplet to spread. DMFBs (Fig. 1b), comprised of a 2D array of individually addressable electrodes, exploit this property to facilitate programmable droplet manipulation. A top plate comprised of a single ground electrode enhances the fidelity of droplet manipulation; dielectric layers insulate the droplet from direct contact with the electrodes. A DMFB may feature (not shown in Fig. 1b) integrated sensors [12] and actuators such as heaters [10] at specific locations on the chips. A host PC or microcontroller facilitates cyber-physical control of the device: control data to electrodes and actuators are sent to the DMFB,

```

1  /* inputs: sample, reagent */
2  analyte = mix sample with reagent for 1m
3  heat analyte at 80c for 2m
4  result = detect analyte for 30s
5  drain analyte

```

Figure 2: A simple assay written in BioScript.

Figure 3: A DMFB compiler must determine when and where operations will be performed, as well as how droplets will arrive at their destinations.

while sensor readings can be processed for real-time analysis and control decisions.

Fig. 1c depicts the basic principle underlying droplet transport, which enables the basic set of operations shown in Fig. 1d. *Input* and *output* operations rely on fluid reservoirs placed on the perimeter of the chip; sensing and heating operations require the presence of additional circuitry and/or external actuators; and, if a camera is present, real-time image acquisition and analysis routines [17] may also be considered as part of the device’s instruction set.

Prior work has demonstrated domain-specific programming languages for DMFBs [1, 6, 14, 20]. This paper leverages one such language, BioScript [14], as a proof-of-concept. Fig. 2 depicts a simple assay written in BioScript. The BioScript syntax allows the programmer to specify the latency of assay operations (e.g., “... for 1m”), but not timing constraints on the intermediate products (fluids) that are generated and later consumed. Using BioScript, the programmer can acquire sensory data from the DMFB, process that data, and make control flow decisions based on the result of the computation; support for functions and arrays is a work-in-progress.

Fig. 3 depicts the portion of a DMFB compiler that is relevant to this paper. The input is a *Directed Acyclic Graph (DAG)*, which represents an assay absent of control flow. The compiler must solve three NP-complete problems: it must schedule assay operations, place scheduled operations on the surface of the chip, and route droplets to and from appropriate locations in accordance with the scheduling and placement result. *This paper adds timing constraints to the scheduling model and introduces new approaches to solve this problem.* Extensions to the compiler to support control flow have been published elsewhere [1], and are fully compatible with the updated scheduling model and algorithms presented here.

3 TIMING CONSTRAINT ANNOTATIONS

This section introduces and characterizes six types of timing constraints on the usage of intermediate fluids that the designers of an assay may want to include in the assay’s specification. To express these constraints, syntactic constructs are presented in the form of

```

1      /* inputs: a, b, c, d, e */
2      @use.in 30s
3      ab = mix 1 units of a with 1 units of b for 15s
4      @finish.at 37s
5      cd = mix 1 units of c with 1 units of d for 10s
6
7      t_e = dispense 2 units of e
8      temp_e = split t_e into 2
9
10     @finish.in 15s
11     heat temp_e[0] at 30c for 15s
12
13     @finish.after 10s
14     heat cd at 90c for 35s
15
16     @use.at 5s
17     cde = mix cd with temp_e[0] for 5s
18
19     @use.after 5s
20     abcde = mix ab with cde for 10s
21
22     abcde = mix abcde with temp_e[1] for 5s
23
24     result = detect sensor on abcde for 5s
25
26     dispose abcde

```

Figure 4: A BioScript assay utilizing all six timing constraint variants.

annotations. As a proof-of-concept, the annotations are added to the syntax of the BioScript language.

3.1 Timing Constraints

Assays that feature time-sensitive reagents must include timing constraints to guarantee correctness. The designer of such an assay must be able to specify these constraints unambiguously in a manner that is both readable by humans and executable or interpretable by machines. Table 1 summarizes six timing constraints that can be applied to an intermediate fluid *produced* by operation u and *consumed* by operation v ; these constraints are associated with the edge (u, v) in the associated DAG.

The constraints in Table 1 are categorized based on how they relate to the consumption of the intermediate fluid; in each constraint Δ is a non-negative value that represents the amount of time specified in the constraint. Constraints starting with S (start) links Δ to the start of the operation v consuming the droplet, while those starting with F (finish) are associated with the completion of operation v . LE , GE , and EQ correspond to the $\leq$, $\geq$ and $=$ comparison operators.

The full set of constraints *can* be expressed by either S or F groupings. For example, an SLE constraint could enforce an FLE constraint by incorporating the latency of v into the value of Δ .

The new syntax in the final column of Table 1 attaches an annotation to an operation u which generates a droplet d ; the associated constraint is then applied to edge (u, v) where v is an operation that consumes d . Split operations may be inserted after u if multiple operations will consume the fluid produced by u .

3.2 BioScript Example

The annotations introduced in the preceding section can be added to any language; we add them to the BioScript language [14] as a proof of concept. BioScript’s type system ensures that

no fluidic variable is used more than once, enforcing the semantic that variable uses are destructive. Fig. 4 shows a toy example assay, written in BioScript and featuring timing annotations: to attach a time constraint to the intermediate fluid (u, v) , an annotation is inserted in the program immediately before operation u .

As described in § 3.1, a timing constraint constrains the duration of time between a pair of operations – e.g., the `@use.in 30s` annotation on line 2 in Fig. 4 indicates there is a SLE constraint on the droplet referred to by ab with $\Delta = 30s$. The annotation associates the constraint with the definition of ab on line 2 and the use of ab on line 20, specifying that operation $abcde = \dots$ on line 20 must *start* within 30 seconds of generating the droplet. *It is the responsibility of the compiler—in particular, of the scheduler—to ensure that this constraint is satisfied.*

4 SCHEDULING PROBLEM

This section reformulates the DMFB scheduling problem to support timing constraints. Using this updated formulation, the following section will present new constraint-aware scheduling algorithms that can be integrated into a compiler. Table 3 lists the notation used throughout this section; Table 4 lists the constraints that characterize a problem instance.

The input is a DAG $G = (V, E)$ and a description of the relevant physical parameters of the DMFB architecture; details about the latter are deferred until § 4.3. Each operation $u \in V$ has a latency $L[u] \in \mathbb{Z}_+$ (Table 2). The scheduler computes the starting time of each operation $S[u]$. The objective is to minimize the latency of the schedule, i.e.,

$$\text{Objective} : \min \left\{ \max_{u \in V} F[u] \right\} \quad (1)$$

where $F[u]$ is the finishing time of operation u :

$$F[u] = S[u] + L[u] \quad (2)$$

4.1 Precedence Constraints

Precedence constraints are fundamental to scheduling. An edge $(u, v) \in E$ means that operation u produces a droplet that operation v will consume; v cannot commence until u completes, i.e.:

$$S[v] \geq F[u], \forall (u, v) \in E. \quad (3)$$

4.2 Fluidic Identifiers (Types)

Scheduling is built on top of a set of identifiers, which we call *types* (not to be confused with *type systems* [14]). Each operation $u \in V$ has a *type* denoted $T[u]$. Likewise, each physical resource in a DMFB can execute assay operations of at least one type.

Each input reservoir on the perimeter of a chip can supply one specific fluid; each output reservoir collects one specific fluid. We treat each uniquely named fluid as its own type. The sets of m and n input and output types are $\mathfrak{I} = \{i_1, \dots, i_m\}$ and $\mathfrak{O} = \{o_1, \dots, o_n\}$.

The electrode array on the DMFB surface performs a set of *reconfigurable* operation types: $\mathfrak{R} = \{split, store, mix, merge\}$ (Fig. 1d); droplet transport is implicit, and is not included as a type. Operations such as *heat* and *sense* that require external modules can be treated as unique types as well.

$\mathfrak{T} = \mathfrak{R} \cup \mathfrak{I} \cup \mathfrak{O} \cup \{heat, sense\}$ is the set of types. $\mathfrak{T}$ can be extended, for example, if a DMFB integrates multiple sensor types.

Table 1: Time constraint variants on the intermediate fluid between operations u and v , which is generated when u completes at time t ; syntax associates a constraint with u , applying it to all (u, v) dependencies

Constraint	Description	New Syntax
SLE Δ	v must begin no later than $t + \Delta$	@use.in Δ
SGE Δ	v must begin no earlier than $t + \Delta$	@use.after Δ
SEQ Δ	v must begin at $t + \Delta$	@use.at Δ
FLE Δ	v must complete no later than $t + \Delta$	@finish.in Δ
FGE Δ	v must complete no earlier than $t + \Delta$	@finish.after Δ
FEQ Δ	v must complete at $t + \Delta$	@finish.at Δ

Table 2: Fluidic operations and their associated latencies

Operation	Latency (in t time-steps)
input	2
output	0
split, merge	2
heat, sense, mix [†]	programmer-specified

[†] Ref [15] relates a mixer's *dimensions* with concrete latencies for homogenization; Ref [9] dynamically adjusts latencies based on a chosen module size.

4.3 DMFB Architecture

Next, we describe our representation of the DMFB architecture. The number of reservoirs (on the perimeter of the chip) of a given input type i_f or output type o_f are denoted as N_{i_f} and N_{o_f} , respectively, where f is the type of the fluid. To ensure that the scheduling problem is tractable, we employ a *virtual topology* (Fig. 5), which partitions the DMFB into a set of N *work modules* based on its dimensions. Absent the virtual topology, the scheduler must effectively solve placement (itself NP-complete) to determine if a schedule is legal; the virtual topology enables schedule legality to be specified via integer linear constraints.

All work modules can execute *reconfigurable* operations of type $\mathfrak{R}$. Work modules overlapping regions of the chip featuring integrated sensors or heaters can additionally perform operations of type *sense* and *heat* respectively; the number of work modules having these capabilities are denoted $N_{sense} \leq N$ and $N_{heat} \leq N$, respectively. A work module can store up to k droplets, depending on its size; at most $N \times k$ droplets may be stored at the same time on the chip.

Figure 5: Virtual topology (VT): to ensure placing and routing feasibility, DMFB resources are partitioned into a set of work modules capable of performing all instruction in Fig. 1d, with interference-free streets reserved for droplet transport. The VT for this 11x8 DMFB exposes 4 work modules, with one capable of heating and one capable of sensing. Interference regions completely surround each work module to prevent inadvertent merging of droplets.

4.4 Resource Constraints

Let $X[j, t]$ be the number operations of type $j \in \mathfrak{T}$ scheduled at time t . The following *resource constraints* must be satisfied at each time-step t :

$$X[i, t] \leq N_i \quad \forall i \in \mathfrak{I} \quad (4)$$

$$X[o, t] \leq N_o \quad \forall o \in \mathfrak{O} \quad (5)$$

$$X[heat, t] \leq N_{heat} \quad (6)$$

$$X[sense, t] \leq N_{sense} \quad (7)$$

$$X[j, t] \leq N * k - 1 \quad \forall j \in \mathfrak{R} \cup \{heat, sense\} \quad (8)$$

Equations (4) to (7) ensure that the number of scheduled operations at time t does not exceed the number of available resources that can execute those operations; Eq. (8) limits droplet storage to prevent deadlock [4].

4.5 Timing Constraints

Let $\mathfrak{C}$ be the set of all timing constraint types listed in Table 1 and let $E_t \subseteq E$ be the set of time-constrained edges. Each edge $(u, v) \in E_t$ is labeled with a constraint pair $\{C[u, v], D[u, v]\}$, where $C[u, v] \in \mathfrak{C}$ is the constraint type and $D[u, v] \in \mathbb{Z}_{\geq 0}$ is the associated duration. Let $SLE, SGE \dots$ be the subsets of edges in E_t that have the associated time constraints. The timing constraints are as follows:

$$S[v] - F[u] \leq D[u, v], \forall (u, v) \in SLE \quad (9)$$

$$S[v] - F[u] \geq D[u, v], \forall (u, v) \in SGE \quad (10)$$

$$S[v] - F[u] = D[u, v], \forall (u, v) \in SEQ \quad (11)$$

$$F[v] - F[u] \leq D[u, v], \forall (u, v) \in FLE \quad (12)$$

$$F[v] - F[u] \geq D[u, v], \forall (u, v) \in FGE \quad (13)$$

$$F[v] - F[u] = D[u, v], \forall (u, v) \in FEQ \quad (14)$$

5 SCHEDULING ALGORITHMS

As noted earlier, the DMFB scheduling problem comprises a DAG $G = (V, E)$ specifying the assay to perform, a subset of edges $E_t \subseteq E$ that feature timing constraints, and a set of parameters characterizing the DMFB architecture (see Table 3). The scheduler must compute the start times $S[u] \forall u \in V$ such that precedence, resource, and timing constraints are satisfied (see Table 4).

Prior work on DMFB scheduling decomposes the schedule on the granularity of time-steps, typically 1 second [18]; assay operations are integer multiples of the time-step. Droplet transport times are assumed to be orders of magnitude faster than assay operations,

Table 3: Notation

	Notation/Variable	Description
Architecture	N	# of work modules the VT partitions the DMFB into
	$N_{heat} \leq N$	# of work modules capable of heating
	$N_{sense} \leq N$	# of work modules capable of sensing
	$N_{if}, i_f \in \mathfrak{I}$	# of input reservoirs with type i_f
	$N_{of}, o_f \in \mathfrak{O}$	# of output reservoirs with type o_f
	k	# of droplets a work module can store
Operations and Types	$G = (V, E)$	DAG: V is the set of operations; E is the set of fluidic dependency edges between operations
	$\mathfrak{R} = \{split, store, mix, merge\}$	Set of reconfigurable operation types
	$\mathfrak{I} = \{i_1, \dots, i_m\}$	Set of input types
	$\mathfrak{O} = \{o_1, \dots, o_n\}$	Set of output types
	$\mathfrak{T} = \mathfrak{R} \cup \mathfrak{I} \cup \mathfrak{O} \cup \{heat, sense\}$	Set of all operation types
	$T[v] \in \mathfrak{T}, v \in V$	Operation type for operation v
	$L[v] \in \mathbb{Z}_+, v \in V$	Latency of operation v
	$S[v] \in \mathbb{Z}_{\geq 0}, v \in V$	Scheduled start time for operation v
	$F[v] \in \mathbb{Z}_+, v \in V$	Scheduled finish time for operation v
	$X[j, t], j \in \mathfrak{T}$	# of scheduled operations of type j at time t
Timing	$E_t \subseteq E$	Subset of fluidic dependency edges having timing constraints
	$\mathfrak{C} = \{SLE, SGE, SEQ, FLE, FGE, FEQ\}$	Set of time constraint types
	$C[u, v] \in \mathfrak{C}, (u, v) \in E_t$	Time constraint type for edge (u, v)
	$D[u, v] \in \mathbb{Z}_{\geq 0}, (u, v) \in E_t$	Time constraint duration for edge (u, v)
	$SLE, SGE, \dots$	The subsets of edges $\in E_t$ having the matching constraint type from $\mathfrak{C}$

Table 4: DMFB Scheduling Problem Constraints

	Constraint	Description
Prec.	$F[u] = S[u] + L[u]$	Operation u 's finish time is its start time + its latency
	$S[v] \geq F[u], \forall (u, v) \in E$	Operation v cannot start until operation u completes
Resource	$X[i_f, t] \leq N_{if}, \forall i_f \in \mathfrak{I}$	No more than N_{if} input operations of type i_f can be scheduled simultaneously
	$X[o_f, t] \leq N_{of}, \forall o_f \in \mathfrak{O}$	No more than N_{of} output operations of type o_f can be scheduled simultaneously
	$X[heat, t] \leq N_{heat}$	No more than N_{heat} heat operations can be scheduled simultaneously
	$X[sense, t] \leq N_{sense}$	No more than N_{sense} sense operations can be scheduled simultaneously
	$X[j, t] \leq N * k - 1, \forall j \in \mathfrak{R} \cup \{heat, sense\}$	One less than the max capacity of all modules can be scheduled simultaneously
Timing	$S[v] - F[u] \geq D[u, v], \forall (u, v) \in SLE$	Operation v must start no later than $D[u, v]$ after operation u completes.
	$S[v] - F[u] \leq D[u, v], \forall (u, v) \in SGE$	Operation v must start no earlier than $D[u, v]$ after operation u completes.
	$S[v] - F[u] = D[u, v], \forall (u, v) \in SEQ$	Operation v must start exactly $D[u, v]$ after operation u completes.
	$F[v] - F[u] \leq D[u, v], \forall (u, v) \in FLE$	Operation v must complete no later than $D[u, v]$ after operation u completes.
	$F[v] - F[u] \geq D[u, v], \forall (u, v) \in FGE$	Operation v must complete no earlier than $D[u, v]$ after operation u completes.
	$F[v] - F[u] = D[u, v], \forall (u, v) \in FEQ$	Operation v must complete exactly $D[u, v]$ after operation u completes.

and are assumed to be zero during scheduling; transport times are computed directly by the droplet router [19], which is beyond the scope of this paper.

This section presents two approaches to our updated variant of the scheduling problem. The first is a heuristic, *Relative Interval Scheduling*, that converges quickly but is not guaranteed to find legal or optimal solutions; the second is an *Integer Linear Program* will compute an optimal schedule, if one exists, but runs in exponential worst-case time unless it is eventually proven that $P = NP$.

5.1 Relative Interval Scheduling

This section presents *Relative Interval Scheduling* (RIS), an efficient heuristic. Limited pseudocode for RIS is available in Appendix A.

RIS proceeds in two phases: the first phase satisfies precedence and timing constraints, and the second satisfies resource constraints. The scheduler fails if either phase cannot satisfy all constraints.

5.1.1 Phase 1 – Satisfying Precedence and Timing Constraints. The first phase of RIS produces an *implicit schedule*, which satisfies precedence (§ 4.1) and timing (§ 4.5) constraints, but not resource constraints (§ 4.4). This phase inserts storage operations into the DAG to ensure $S[v] = F[u], \forall (u, v) \in E$, ensuring that each droplet is either used or stored immediately after it is produced.

For each edge $(u, v) \in E_t$ with constraint type $C[u, v]$ and duration $D[u, v]$ (see § 4.5), we consider three cases:

1 – Infeasible Edges: If $C[u, v] \in \{FLE, FEQ\} \wedge D[u, v] < L[v]$, then the constraint cannot be satisfied; no legal schedule exists.

2 – Storage Node Insertion: If $S[v] = F[u]$ violates a timing constraint, then a storage operation s of appropriate duration is inserted into the DAG between u and v .

3 – Storage Windows: Let δ be the maximum value that can satisfy the constraint should $S[v] = F[u] + \delta$. To model this possibility, we associate a *storage window* $W[u, v] = \delta$ with edge (u, v) indicating that we can store the droplet produced by u for up to δ time-steps while satisfying the timing constraint. Additionally, we set $W[u, v] = \infty$ for each unconstrained edge $(u, v) \in E \setminus E_t$.

Storage Window Expansion: Storage operations can be inserted along DAG edges where storage windows exist. Consider an edge $(u, v) \in E$ with storage window $W[u, v] = \delta > 0$. We can *expand* $W[u, v]$ by inserting a storage operation s with latency $\lambda \leq \delta$, and then reducing $W[u, v]$ by λ . If $W[u, v]$ remains positive, then subsequent expansions may be possible; with these limits, storage window expansion cannot induce new timing constraint violations.

Path Length Equalization: A path P in G is a sequence of vertices that are connected by edges. The latency of P , denoted $L[P]$, is the sum of the latencies of the vertices on the path. Two paths P_1 and P_2 in a DAG are *divergent-convergent* if they share the same initial vertex u and terminal vertex v , and no other vertices. Without loss of generality, if $L[P_1] < L[P_2]$, then one or more storage nodes totalling $L[P_2] - L[P_1]$ time-steps will need to be added to P_1 ; this ensures that all droplets consumed by v will be present at time-step $S[v]$. This can be accomplished by applying storage window expansion on vertices in P_1 , stopping when $L[P_1] = L[P_2]$. If it is not possible to do so, then a legal schedule does not exist. Equalizing the lengths of all divergent-convergent path pairs ensures that $S[v] = F[u], \forall (u, v) \in E$. We integrate path-length equalization with a call to an algorithm that computes a latency-constrained schedule of the DAG, such as *As Soon As Possible (ASAP)* or *As Late As Possible (ALAP)* scheduling [11]; details are omitted to conserve space. We abstract this process as a function $\Omega(G)$ which returns an implicit schedule of operations in G , starting at time 0.

5.1.2 Relative Interval Forest. The *relative interval forest* is a data structure that encodes the implicit schedule corresponding to $\Omega(G)$ in a way that provides an efficient mechanism to test for resource constraint violations at any time-step of the schedule.

Droplet Lifetime: The lifetime of droplet d can be characterized by a path $P_d = \langle u_1, u_2, \dots, u_m \rangle$ where u_1 is the operation that produces d and u_m is the operation that consumes d . The operations that produce new droplets are inputs, mixes, merges, and the immediate successors of splits; the operations that consume droplets are outputs, splits, and the immediate predecessors of mixes or merges. Sensing and heating operations do not consume or produce new droplets; a sensing operation produces data from a droplet, while a heating operation changes a physical property of the droplet.

Relative Interval Tree: Path length equalization ensures that $S[u_{i+1}] = F[u_i], 1 \leq i \leq m - 1$. In other words, the lifetimes of the operations that comprise the lifetime of d are a set of contiguous intervals $[S[u_1], F[u_1]][F[u_1], F[u_2]] \dots [F[u_{m-1}], F[u_m]]$ with no gaps between them. To evaluate resource constraints, the scheduler may ask if an operation of type j is processing droplet d at time t . The answer is ‘yes’ if there exists an operation $u_i \in P_d$ such that

$S[u_i] \leq t < F[u_i] \wedge T[u_i] = j$ and ‘no’ otherwise. This query can be answered in $O(m)$ time using a linear search of the intervals, or in $O(\log m)$ time by representing the intervals with a balanced binary search tree, which we call a *relative interval tree*. We employ the latter, omitting details to conserve space.

A relative interval forest consists of a set of relative interval trees for all of the variables in a DAG. A *query* $Q[j, t]$ on a relative interval forest returns the set of operations of type j scheduled to execute at time t , i.e.,

$$Q[j, t] = \{u \in V \mid S[u] \leq t < F[u] \wedge T[u] = j\}. \quad (15)$$

By querying the relative interval forest, it is straightforward to determine whether or not any of the resource constraints listed in Table 4 are violated at time t .

5.1.3 Phase 2 – Satisfying Resource Constraints. Phase 2 of RIS tries to resolve any resource constraint violations present in the implicit schedule $\Omega(G)$. It first resolves all resource constraint violations in each connected component $\gamma \in G$, as described next, then checks for violations when all γ are considered. Violations existing across connected components are trivially resolved by shifting components’ schedules in relation to each other; in the worst case, components are scheduled in a way where none of them overlap, as timing constraints and fluidic dependencies are absent between them. $\Omega(G)$ returns when all constraints are satisfied.

Resolving Resource Violations: Without loss of generality, suppose that one of Eqs. (4) to (7) is not satisfied in a connected component. Let $\langle t_i, t_{i+1}, \dots, t_{i+m} \rangle$ be the maximum-length contiguous sequence of time-steps where resource constraints for type j are not satisfied, i.e., $X[j, t_\alpha] > N_j, i \leq \alpha \leq i + m$. We say that a resource violation of duration m exists between the set of droplets Θ that correspond to operations returned by queries $Q[j, t_\alpha], i \leq \alpha \leq i + m$.

The basic strategy is to expand storage windows between some of the operations in the respective lifetimes of some of the droplets in Θ , adjusting the schedule in a manner that satisfies one or more resource constraints (i.e., reducing the value of $X[j, t_\alpha]$; however, doing so may introduce new resource violations in other parts of the DAG. Whenever a storage window is expanded, RIS recomputes the relative interval forest, iteratively lengthening the schedule until all resource constraints are satisfied. For each distinct pair of droplets $\{d_1, d_2\} \in \Theta$, we expand storage windows along particular paths in the DAG to offset their relative positions by m ; there are three cases to consider:

1 – Convergent (or Divergent) Paths: If d_1 and d_2 are on paths that converge (or diverge), we expand storage windows that exist *after* (or *prior* to) the operations involved in the violation.

2 – Divergent-Convergent Paths: If d_1 and d_2 are on a pair of divergent-convergent paths, we expand storage windows by an equal amount on opposite sides of the offending operations.

3 – Descendent-Ancestor Paths: If d_1 and d_2 have descendants with a common ancestor $\hat{u}_a$ that is not an ancestor of the operations that produce d_1 and/or d_2 , then we expand either of the paths between $\hat{u}_a$ and the associated descendent.

If a violation of Eq. (8) exists within any connected component, where all droplets are related by the first two cases described above, then RIS aborts; in this case, expanding any storage windows would

Figure 6: Relative Interval Scheduling (RIS): (a) is parsed by RIS in Phase I to impose a schedule satisfying all timing constraints; given the architecture in Fig. 5, the Relative Interval Forest (b) corresponding to the schedule found in Phase I exposes a resource violation from [39, 49). During Phase II, RIS expands the storage window occurring at [54, +inf), resolving this violation.

not reduce the number of simultaneously scheduled droplets below the threshold.

5.1.4 RIS Example. Fig. 6a depicts the DAG corresponding to the BioScript assay shown in Fig. 4. Phase 1 of RIS inserts storage operations and storage windows in the DAG, leading to the construction of the relative interval forest on $\Omega(G)$ shown in Fig. 6b. For example, the *finish.after* constraint with $\delta = D[u, v] = 10s$ on the edge (u, v) between the *heat* (u) of variable cd and subsequent *mix* (v) of cd and e requires that $F[v] \geq F[u] + \delta$ (§ 3.1). As $L[v] = 5 \not\geq \delta$, phase one inserts a storage operation s into G with a latency of $\delta - L[v] = 5$ as described in § 5.1.1. Additionally, as the constraint allows v to finish at *any* time later than $F[u] + \delta$, storage window $W[s, v] = \infty$ is associated on (s, v) . Phase 2 of RIS then finds the resource violation of the overlapping *heat* operations over the interval $[39, 49)$ shown in Fig. 6b and expands $W[s, v]$ inserted in phase 1 to resolve this constraint (Fig. 6c). In this state, the forest reveals that no resource violations exist; hence, $\Omega(G)$ is a legal schedule of G on the given architecture.

5.2 Integer Linear Programming Formulation

We present an *Integer Linear Programming (ILP)* formulation of the DMFB scheduling problem with timing constraints; an ILP solver can produce optimal solutions, but will not be able to scale to large instances. The formulation derives from prior work ([13]) which does not support timing constraints.

The ILP scheduler first computes an upper bound B on the length of the schedule using any efficient (and presumably sub-optimal) heuristic [13, 18]. For each $u \in V$ and $1 \leq t \leq B$, we introduce a binary variable $x_{u,t}$ which is set to 1 if u is scheduled to start at time t and 0 otherwise (Eq. (16)); the solver ensures that each operation is scheduled to start at exactly one time-step (Eq. (17)) and determines the start time $S[u]$ of each operation (Eq. (18)):

$$x_{u,t} = \begin{cases} 1, & \text{if operation } u \text{ starts at time-step } t \\ 0, & \text{otherwise} \end{cases} \quad (16)$$

$$\sum_{t=1}^B x_{u,t} = 1, \forall u \in V \quad (17)$$

$$S[u] = t | x_{u,t} = 1 \quad (18)$$

The scheduling objective, operation finish times, and precedence constraints are obtained directly from Eqs. (1) to (3).

Resource constraints from Eqs. (4) to (8) are obtained directly after first counting the number of operations of each type that are scheduled at each timestep; we compute each count as follows:

$$X[\tau, t] = \sum_{u \in v | T[u] = \tau} \sum_{j=\max\{1, t-L[u]\}}^t x_{u,j}, \forall \tau \in \mathfrak{T} \quad (19)$$

Storage operations are inferred from the schedule. For each edge $(u, v) \in E$, variable $z_{u,v,t}$ is set to 1 if the corresponding droplet is

stored during time-step t , $S[v] > t > F[u]$ and 0 otherwise.

$$z_{u,v,t} = \sum_{k=1}^{t-L(u)} x_{u,k} - \sum_{k=0}^t x_{v,k} \quad \forall (u, v) \in E \quad (20)$$

$$X[\text{store}, t] = \sum_{(u,v) \in E} z_{u,v,t} \quad 1 \leq t \leq B \quad (21)$$

Timing constraints described previously in Eqs. (9) to (14) are modeled directly for all $(u, v) \in E_t$.

We observed that the ILP-based scheduler led to an inordinate number of seemingly unnecessary storage operations, which increased the amount of work being done by the placer and router downstream. To compensate, we introduced a secondary scheduling objective to minimize the amount of allocated storage:

$$\text{Minimize : } \sum X[\text{store}, t], 1 \leq t \leq B \quad (22)$$

6 BENCHMARKS

To the best of our knowledge, benchmarks that presently exist for evaluation of microfluidic compilers and synthesis tools do not feature timing constraints akin to those introduced in this paper. We created two sets of benchmarks by adding timing constraints to existing benchmarks, all of which are DAGs. The original (unmodified) benchmarks have been widely used in prior papers that evaluate DMFB scheduling [13] without timing constraints.

SLE-only consists of DAGs annotated with SLE constraints, serving to limit the amount of time that a volatile reagent could be stored before spoilage occurs; all SLE constraints were set with $\Delta = 0$ to maximally stress the schedulers. *SLE-only* is further separated into three sections: the first (unnamed) section contains *ELISA*, an immunoassay that detects various opiates in a sample, and the *Polymerase Chain Reaction (PCR)*, which amplifies DNA through replication. The *Multiplexed* section contains a multiplexed version of PCR (with 4 targets to replicate), as well as 5 variants of an *in vitro* diagnostics assay, where all combinations of the number of samples (s) and reagents (r) are evaluated. Lastly, the *ProteinSplit* assays in the *Split-Dilutes* section employ high fan-out trees to dilute a sample to varying concentrations.

Mixed category DAGs were synthetically generated to stress-test the scheduler's ability to satisfy different types of timing constraints. The two versions of *all_six* utilize all six timing constraints introduced in this paper, while *all_eq*, *all_finish* and *all_start* use only the types of constraints that match their names. The last benchmark has an *infeasible* set of constraints; no legal schedule exists.

Benchmark specifications and associated DAGs are [here](#).

7 EVALUATION

7.1 Setup

All experiments were performed on a mobile PC with an Intel® Core™ i9 processor, with 1TB SSD storage and 32GB DDR4 memory running macOS 10.15.6®. All assays were written in BioScript [14] using the constraint annotations introduced in § 3. Each assay was compiled and passed to an open source cycle-accurate DMFB synthesis simulation framework [5] for evaluation.

We compare against four baseline schedulers available within the framework that do not account for timing constraints: **List**

Table 5: Scheduling Results and Execution Time for Each of the Scheduling Algorithms Described in § 6

Suite	Benchmark	Scheduled Time (seconds)						Execution Time (milliseconds)					
		LS	FDLS	PS	RIS	ILP	ILP-T	LS	FDLS	PS	RIS	ILP	ILP-T
SLE-only	ELISA	Fail (1)	Fail (1)	Fail (1)	<u>7,600</u>	Fail (1)	<u>7,600</u>	-	-	-	<u>48</u>	-	333,233
	PCR	Fail (1)	Fail (1)	Fail (1)	<u>697</u>	Fail (1)	<u>697</u>	-	-	-	<u>4</u>	-	1,220
	Multiplexed	M-PCR	<u>1,042</u>	<u>1,042</u>	<u>1,042</u>	<u>1,042</u>	<u>1,042</u>	<u>14</u>	<u>11,582</u>	<u>14</u>	<u>22</u>	<u>133,436</u>	<u>104,498</u>
		InVitro 2s,2r	<u>15</u>	<u>15</u>	<u>15</u>	17	<u>15</u>	<u>2</u>	<u>3</u>	<u>1</u>	3	<u>5,131</u>	<u>5,158</u>
		InVitro 2s,3r	19	19	19	19	<u>18</u>	<u>3</u>	<u>6</u>	<u>3</u>	<u>6</u>	<u>5,210</u>	<u>5,304</u>
		InVitro 3s,3r	Fail (5)	Fail (4)	Fail (3)	24	Fail (1)	-	-	-	<u>17</u>	-	<u>6,005</u>
		InVitro 3s,4r	Fail (9)	Fail (7)	Fail (1)	27	Fail (3)	-	-	-	<u>36</u>	-	<u>7,546</u>
		InVitro 4s,4r	Fail (10)	Fail (7)	Fail (2)	32	Fail (1)	-	-	-	<u>77</u>	-	<u>10,520</u>
	Split-Dilutes	ProteinSplit1	Fail (6)	Fail (6)	64	<u>55</u>	Fail (2)	-	-	<u>3</u>	<u>23</u>	-	<u>6,057</u>
		ProteinSplit2	Fail (12)	Fail (12)	91	Fail (R)	Fail (7)	-	-	<u>3</u>	-	-	<u>31,902</u>
		ProteinSplit3	Fail (24)	Fail (24)	145	Fail (R)	Fail (20)	-	-	<u>12</u>	-	-	<u>322,721</u>
		ProteinSplit4	Fail (48)	Fail (48)	<u>253</u>	Fail (R)	Fail (56)	-	-	<u>6</u>	-	-	-
		ProteinSplit5	Fail (96)	Fail (96)	<u>562</u>	Fail (R)	Fail (T)	-	-	<u>14</u>	-	-	-
	Mixed	all_six	Fail (6)	Fail (6)	Fail (6)	<u>100</u>	Fail (5)	-	-	-	<u>5</u>	-	<u>14,692</u>
		all_six_2	Fail (4)	Fail (4)	Fail (4)	<u>237</u>	Fail (4)	-	-	-	<u>2</u>	-	<u>11,881</u>
		all_eq	Fail (7)	Fail (7)	Fail (8)	<u>64</u>	Fail (7)	-	-	-	<u>4</u>	-	<u>5,860</u>
		all_finish	Fail (2)	Fail (2)	Fail (2)	<u>38</u>	Fail (2)	-	-	-	<u>1</u>	-	<u>4,578</u>
		all_start	Fail (2)	Fail (2)	Fail (2)	<u>64</u>	Fail (2)	-	-	-	<u>2</u>	-	<u>4,352</u>
		infeasible	Fail (1)	Fail (1)	Fail (1)	Abort	Fail (1)	-	-	-	<u>1</u>	-	<u>150</u>
Fail:	(T) ILP-T timed out without improving upon or confirming optimality of seeded solution												
	(# number of timing constraints violated)												
	(R)												
bold	best rest												
	<u>underline</u>												
Abort:	scheduler reported infeasibility and aborted												
	-												
	RIS failed to overcome resource constraints												
	optimal schedule												
	time not reported due to failure												

Scheduling (LS) [18], **Force-directed List Scheduling (FDLS)** [13], **Path Scheduling (PS)** [13], and an **ILP**-based scheduler [13, 18], which we rewrote to use the Gurobi optimizer¹. We implemented **Relative Interval Scheduling (RIS)** within the same framework and extended the ILP scheduler to include timing constraints (**ILP-T**).

For *SLE-only*, we target a 15×19 DMFB architecture featuring input ports with a 2-second latency, output ports with zero latency, and 5 heating and sensing regions available. For *Mixed*, we target the same DMFB with a single heating region, similar to the scheduling example with resource constraint violation in Fig. 6.

Table 5 reports the length of each schedule (in seconds) along with the execution time of each algorithm. For each benchmark, the shortest obtained schedule that satisfies all constraints is reported in bold; optimal solutions, as verified by the ILP solver, are underlined. Table 5 also reports scheduling failures:

- The baseline heuristics and unmodified ILP may find schedules that satisfy resource and precedence constraints, but not timing constraints; we report the number of timing constraint violations as *Fail*(#)
- RIS may satisfy precedence and timing constraints, but not resource constraints; when this occurs, we report *Fail*(R)
- We give ILP and ILP-T a 4-hour timeout; if a legal schedule is not found within 4 hours, we report *Fail*(T)

¹<https://www.gurobi.com>

We only report the execution time of successful scheduling runs; if a scheduler is able to determine that no legal schedule can be found that satisfies all timing constraints, it aborts and reports “infeasibility” which we consider to be an optimal result.

7.2 Simulation Results: Schedule Length

LS, **FDLS**, and **ILP** found legal schedules for 3 of the 19 benchmarks, but failed to satisfy at least one timing constraint for the others. On average, ILP generated schedules with 23% and 22% fewer timing constraint violations than LS or FDLS. Results for LS and FDLS only differed for the 3 largest multiplexed *In vitro* benchmarks, with FDLS having 24% fewer violations for them.

PS successfully scheduled 8 of the 19 benchmarks; it was more successful than FS, FDLS, and ILP, especially in the *Split-Dilutes* subcategory, noting that PS was optimized for DAGs with high fan-out [13]. ILP-T confirmed that PS’ schedules for ProteinSplit1-3 were sub-optimal, but timed out for ProteinSplit4-5.

RIS succeeded in 15 of the 19 benchmarks, 10 of which were confirmed as optimal. Notably, RIS produced a shorter schedule than PS for ProteinSplit1; however, its four failures were for the larger ProteinSplit2-5 DAGs, which are all trees with high fan-out. The failures occurred as a byproduct of the process for choosing storage windows to expand when resource violations arise; determining better expansion orders is left open for future work.

ILP-T reported the optimal time for 17 of the 19 benchmarks; the two failures are due to timeout after 4 hours. In 5 of the 17 successful cases, ILP-T reported shorter schedules than the best result obtained among the heuristics.

7.3 Execution Time

The four heuristics ran orders of magnitude faster than ILP or ILP-T. Among the heuristics, RIS was a bit slower than LS or PS. The runtime of RIS was comparable to FDLS for all but one benchmark (M-PCR), where FDLS ran orders of magnitude slower (but still much faster than ILP or ILP-T). For the three benchmarks where ILP and ILP-T found legal solutions, their runtimes were comparable for two, while ILP ran a bit slower than ILP-T for the third (once again, M-PCR).

If a legal schedule that satisfies timing constraints is desired, a good strategy might first try scheduling using RIS, followed by scheduling using ILP-T. The good-quality schedules produced by RIS will provide a reasonable lower-bound on the schedules that can be achieved.

8 RELATED WORK

Prior work has studied the DMFB scheduling problem with precedence and resource constraints, but not timing constraints [3, 13, 18]. Most of these methods were either ILPs [3, 13, 18] or constructive heuristics [13, 18] that added one vertex or path at a time to a partial schedule, while ensuring that resource constraints were always satisfied; iterative improvement metaheuristics [18] repeatedly call constructive heuristics as well. The Relative Interval Scheduling mechanism presented here takes a different approach: starting with a schedule that does not satisfy timing or resource constraints, it incrementally refines the schedule until it does. The ILP presented here is similar to prior work, with the inclusion timing constraints.

Ref. [7] considered timing constraints for fluidic routing in channel-based microfluidic chips featuring very large scale integration of software-programmable microvalves. These chips employ external syringe pumps and solenoid valves to drive fluid flow, dynamically reconfiguring fluid routing paths at runtime. For this problem, the schedule of operations is given, and the objective is to physically lay out the channel network to ensure that timing constraints provided by the schedule are not violated. In contrast, the approach presented here satisfies timing constraints during scheduling, without considering the implications on placement and routing, as fluid transport is significantly faster than assay operation latencies in DMFB technology.

9 CONCLUSION AND FUTURE WORK

This paper identified and provided solutions to an important problem for writing and executing assays on programmable microfluidic technologies; the key insight is that most real world assays have timing-sensitive reactions — whether listed explicitly or implied — which are unable to be specified or enforced using existing languages and compilers. Any language and compiler could adopt the methods presented here to enable fine-tuned control over timing-constrained operations at the granularity of any straight-line program with no branches or branch targets (i.e., a basic block). Future

work should investigate precise methods to schedule programs satisfying time constraints across multiple execution paths.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1351115, Grant No. 1545097, Grant No. 1740052, Grant No. 1910878, and Grant No. 2019362.

REFERENCES

- [1] 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, Vienna, Austria, February 24–28, 2018*. ACM, 365–377.
- [2] Huijiang Ding, Saman Sadeghi, Gaurav J Shah, Supin Chen, Pei Yui Keng, R Michael van Dam, et al. 2012. Accurate dispensing of volatile reagents on demand for chemical reactions in EWOD chips. *Lab on a Chip* 12, 18 (2012), 3331–3340.
- [3] Jie Ding, Krishnendu Chakrabarty, and Richard B. Fair. 2001. Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 20, 12 (2001), 1463–1468.
- [4] Daniel Grissom and Philip Brisk. 2012. Fast Online Synthesis of Generally Programmable Digital Microfluidic Biochips. In *Proceedings of the Eighth IEEE/ACM/FIP International Conference on Hardware/Software Codesign and System Synthesis (Tampere, Finland) (CODES+ISSS '12)*. ACM, 413–422.
- [5] 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.
- [6] Daniel T. Grissom, Christopher Curtis, and Philip Brisk. 2014. Interpreting Assays with Control Flow on Digital Microfluidic Biochips. *ACM J. Emerg. Technol. Comput. Syst.* 10, 3 (2014), 24:1–24:30.
- [7] Xing Huang, Tsung-Yi Ho, Krishnendu Chakrabarty, and Wenzhong Guo. 2019. Timing-driven flow-channel network construction for continuous-flow microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 6 (2019), 1314–1327.
- [8] Gabriel Lippmann. 1875. *Relations entre les phénomènes électriques et capillaires*. Ph.D. Dissertation. Gauthier-Villars Paris, France.
- [9] Tyson Loveless, Jason Ott, and Philip Brisk. 2020. A performance-optimizing compiler for cyber-physical digital microfluidic biochips. In *CGO '20: 18th ACM/IEEE International Symposium on Code Generation and Optimization*. ACM, 171–184.
- [10] 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.
- [11] Giovanni De Micheli. 1994. *Synthesis and Optimization of Digital Circuits* (1st ed.). McGraw-Hill Higher Education.
- [12] Miguel Angel Murran and Homayoun Najjaran. 2012. Capacitance-based droplet position estimator for digital microfluidic devices. *Lab Chip* 12 (2012), 2053–2059. Issue 11.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] Veronica Sanchez-Freire, Antje D Ebert, Tomer Kalisky, Stephen R Quake, and Joseph C Wu. 2012. Microfluidic single-cell real-time PCR for comparative analysis of gene expression patterns. *Nature protocols* 7, 5 (2012), 829.
- [17] 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.
- [18] Fei Su and Krishnendu Chakrabarty. 2008. High-level synthesis of digital microfluidic biochips. *JETC* 3, 4 (2008), 1.
- [19] Fei Su, William L. Hwang, and Krishnendu Chakrabarty. 2006. Droplet routing in the synthesis of digital microfluidic biochips. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6–10, 2006*. IEEE, 323–328.
- [20] Max Wilsey, 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-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019*. ACM, 183–197.

A PSEUDOCODE FOR RELATIVE INTERVAL SCHEDULING

Relative Interval Scheduling (§ 5.1) is summarized in Algorithm 1. While the procedure is relatively straightforward, partial pseudocode is provided for parsing time constraints in phase 1 (Algorithm 2), and for expanding storage windows in phase 2 (Algorithm 3).

Algorithm 1 Relative Interval Scheduling

```

1: procedure SCHEDULE(Architecture A, DAG G)
2:   ParseTimeConstraints(G) ▷ aborts if infeasible
3:   forest  $\leftarrow$  CreateForest(G)
4:   violations  $\leftarrow$  GetViolations(forest, A) ▷ fails if not enough modules
5:   while violations  $\neq \emptyset$  do
6:     ExpandStorageWindow(G, violations) ▷ fails if no progress
7:     forest  $\leftarrow$  CreateForest(G)
8:     violations  $\leftarrow$  GetViolations(forest, A)
9:   return  $\Omega(G)$ 

```

Algorithm 2 Parsing Time Constraints (Phase 1)

```

1: procedure PARSETIMECONSTRAINTS(DAG G)
2:   for child  $\in$  reverse(G) do
3:     for parent  $\in$  parents(child) do
4:       if isConstrained(parent) then
5:         clat  $\leftarrow$  child.latency
6:         dur  $\leftarrow$  parent.constrDuration
7:         newStore  $\leftarrow$  StorageNode
8:         switch parent.constrType do
9:           case SGE
10:            newStore.window  $\leftarrow$  Window(+inf)
11:           case SEQ
12:            newStore.latency  $\leftarrow$  dur
13:            G.insertStoreBetween(newStore, parent, child)
14:            break
15:           case SLE
16:            parent.window  $\leftarrow$  Window(dur)
17:            break
18:           case FGE
19:            newStore.window  $\leftarrow$  Window(+inf)
20:           case FEQ
21:            if dur - clat < 0 then
22:              Abort(infeasible)
23:            newStore.latency  $\leftarrow$  dur - clat
24:            G.insertStoreBetween(newStore, parent, child)
25:            break
26:           case FLE
27:            if dur - clat < 0 then
28:              Abort(infeasible)
29:            parent.window  $\leftarrow$  Window(dur - clat)
30:            break
31:           else
32:             parent.window  $\leftarrow$  Window(+inf)
33:   EqualizePaths(G) ▷ omitted

```

Algorithm 3 Satisfy a Resource Violation

```

1: procedure EXPANDSTORAGEWINDOW(DAG G, Set of Violations V)
2:   for violation in V do
3:     dur  $\leftarrow$  violation.duration
4:     for pair of ops  $u, v$  in violation.overlappingOps do
5:       switch path relationship between  $path_u$  and  $path_v$  do
6:         case  $path_u$  and  $path_v$  converge at  $w$ 
7:           if Storage windows exist between  $u$  and  $w$  with
             combined durations  $\geq$  dur then
8:             expanded  $\leftarrow$  0
9:             for window  $\omega$  between  $u$  and  $w$  do
10:              if expanded = dur then
11:                break
12:              expand  $\omega$  by  $\Delta = \min\{dur -$ 
13:                expanded,  $\omega.size\}$ 
14:              expanded  $\leftarrow$  expanded +  $\Delta$ 
15:            return
16:            case  $path_u$  and  $path_v$  diverge from  $w$ 
17:              ...
18:            case  $path_u$  and  $path_v$  diverge from  $y$  and converge at
               $z$ 
19:              ...
20:            case  $path_u$  and  $path_v$  neither converge nor diverge
21:              ...
21:   Fail(|V|) ▷ did not make any progress

```