A High-Performance Online Assay Interpreter for Digital Microfluidic Biochips

Daniel Grissom, Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
{grissomd, philip}@cs.ucr.edu

ABSTRACT

We introduce an online interpreter to execute biochemical assays on droplet-based digital microfluidic biochips (DMFBs). Online interpretation enables adaptivity, e.g., response to faults during assay execution, variable-latency assay operations, and concurrent workloads whose composition is not known statically. Our online method routes droplets dynamically, making decisions in milliseconds while running on a low-cost Intel Atom™ processor.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; J.3 [**Life and Medical Sciences**]: Biology and Genetics, Health

General Terms

Algorithms, Design, Performance

Keywords

Digital Microfluidic Biochip (DMFB), Electrowetting-on-Dielectric (EWoD), Virtual Architecture, Routing, Deadlock.

1. INTRODUCTION

Digital microfluidic biochips (DMFBs) are cyber-physical MEMS devices that manipulate droplets of liquid on a 2-dimensional grid (Fig. 1). A DMFB is a planar array of indistinguishable *cells*; a cell is the abstraction of a square region on top of each electrode. Cells can perform basic operations—e.g., droplet movement, merging, mixing, splitting, storage—that form building blocks for larger chemical reactions called *assays*.

DMFBs will pave the way for programmable chemistry: the chemist of the future will specify assays using domain-specific languages. The assay representation is compiled into a sequence of *droplet actuation cycles*. Each cycle specifies a set of signals to be sent to the DMFB to actuate droplet movement. A program running on a computing device connected to the DMFB traverses the sequence of cycles, sends the appropriate signals for each cycle, and holds the signals for an appropriate period of time to ensure that all droplets complete their movements.

Historically, DMFB compilation has been performed offline. This works fine under ideal circumstances in which the behavior of the system can be characterized statically, and execution proceeds without error; however, it cannot handle any form of variability.

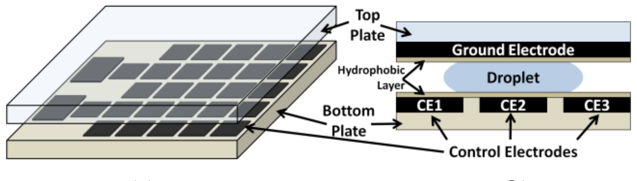

Figure 1. (a) DMFB with a 2D array of control electrodes; (b) DMFB cross-section: a droplet is centered on top of electrode CE2 and overlaps adjacent electrodes, CE1 and CE3. A voltage applied to CE1 or CE3 induces motion to the left or right. Other feasible operations include droplet splitting, merging, mixing, and storage.

This paper introduces online assay interpretation for DMFBs, as an alternative to static compilation. Online interpretation will enable interesting new capabilities in the areas of control flow and dynamic scheduling. For instance, a particular assay could be executed based on the results of a previous assay or environmental condition. Other potential benefits include the ability to detect and respond dynamically to faults or to dynamically adjust a schedule to account for variable-latency operations [2].

Static compilation methods employ long-running algorithms that produce highly optimized results, effectively minimizing assay completion time. Online interpretation, in contrast, must overlap algorithmic decision-making steps with the execution of each cycle. A DMFB typically runs at around 100 Hz, meaning that each cycle lasts for around 10ms. If the online decision-making algorithm runs for longer than 10ms, then the length of the cycle must be extended, thereby increasing assay execution time.

A secondary consideration is the cost of the computing device that is connected to the DMFB, especially when integrated into products that perform low-cost portable point-of-care diagnostics. Ideally, 10ms runtimes could be achieved on low-cost battery-operated embedded computers, as opposed to higher-performance power-hungry desktop PCs. The interpreter described here meets these constraints on a single-threaded Intel Atom™ processor.

2. RELATED WORK

Static DMFB compilers must solve three NP-complete problems: scheduling [7], placement [8], and routing [1, 3, 5, 9]. High quality solutions are achieved using long-running iterative improvement algorithms [7, 8] or optimally via Integer Linear Programming [7] or A* Search [1].

Obviously, these methods do not meet the real-time constraints imposed by the interpretation framework. For example, the BioRoute router reports droplet routing times as low as 40ms on a 1.2 GHz Sun Blade-2000 machine with 8GB of memory. Our interpreter, in contrast, is able to meet the timing constraints while running on a low-cost Intel Atom™ processor.

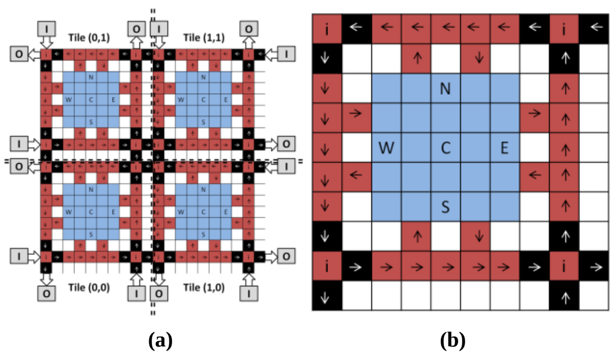

Figure 2. (a) A 2x2 virtual architecture represented as an array of tiles (separated by double, dashed, black lines) showing potential I/O locations; (b) A tile, comprised of a work chamber ('C'), 4 streets, and 4 intersections ('i').

Griffith and Akella [5] impose a *virtual architecture* on the DMFB that restricts the functions that different cells can perform. Similar to the layout of a city, the DMFB is organized into streets, intersections, *rotaries* (e.g., traffic circles), and city blocks (where all assay operations happen). This limits the flexibility of the DMFB, but facilitates the adaptation of network routing algorithms for droplet transport. To prevent deadlock, they limit the injection rate of droplets into the system. Our interpreter takes a similar approach, but adopts mesh network deadlock-free routing algorithms [4], rather than limiting the injection rate.

3. VIRTUAL ARCHITECTURE

As shown in Fig. 2, our interpreter imposes a virtual architecture onto the DMFB, and exploits its restrictive structure to achieve fast algorithmic runtimes. Input and output reservoirs are placed on the perimeter of the DMFB.

The city blocks are referred to as (reaction) *chambers*, because all assay operations occur there. Each chamber can perform one operation (e.g., merging, mixing or splitting) or can store up to four droplets. External devices such as heaters or optical detectors can be affixed to the DMFB above or below a chamber.

All streets are 1-way; eight 1-one streets come together at rotaries (Fig. 2(a)), which offer an abstraction like a network router. Droplets travel clockwise through the rotary. Without loss of generality, a droplet traveling north that enters a rotary could continue straight, or turn left (west) or right (east); our routing algorithms do not allow droplets to reverse directions, so a droplet would not enter a rotary traveling north and then exit traveling south.

Each chamber and the four adjacent streets surrounding it is called a *tile*, as shown in Fig. 2(b). For a 5x5 chamber size, each tile is a 10x10 array of cells. Tiles are repeated to form a virtual architecture, e.g., Fig. 2(a) shows a 2x2 array of tiles.

3.1 Simplifying Placement and Routing

The virtual architecture simplifies the placement and routing steps of an assay as follows:

Placement: we sacrifice the ability to place an assay operation at any DMFB location. Instead, the interpreter dynamically *binds* operations to chambers.

Routing: traditional routers move droplets across the DMFB in a chaotic and unorderly fashion while ensuring separation between all droplets at all times. The DMFB imposes a city-like network of streets that all droplets must follow. This limits the number of routes between each source-destination, which simplifies routing.

3.2 Intermediate Bytecode Language

Conceptually, the set of signals sent to a DMFB during each cycle can be treated like a machine language. If the DMFB is comprised of *N* cells, then *N* binary signals are sent to the device (e.g., a '1' activates an electrode, and a '0' leaves it off).

The virtual architecture is a virtual machine with its own *intermediate bytecode language* that operates at a higher-level of abstraction than the DMFB machine language described above. There are two types of bytecode instructions: operations (*O-type*) and transport (*T-type*):

Each O-type instruction has the form *(opcode, chamber-id)*, where the opcode specifies the operation to perform, and the chamber-id specifies which chamber to perform the operation. All chambers support four basic opcodes: *{start-mix, stop-mix, split}*. If a chamber has an external device affixed to the outside of the chip, such as a heater or detector, then it may support additional opcodes such as *{heater-on, heater-off, detector-on, detector-off}*.

Each T-Type instruction has the form *(droplet-id, src, dst)*; the droplet-id specifies a droplet originating at the source (src), and the instruction is to transport the droplet to the destination (dst). The source may be a chamber or an input reservoir, and the destination may be a chamber or an output or waste reservoir. The droplet-id field is necessary to handle the situation where a chamber is storing multiple droplets, but can be dropped when convenient: an input reservoir generates a new droplet, so no droplet-id field is necessary; similarly, if the chamber only contains one droplet, then its id is implicitly known. If a T-type instruction transports a droplet to a chamber, it is stored implicitly, until an O-type instruction initiates an operation.

4. ONLINE INTERPRETER

Assays are specified as directed acyclic precedence graphs (DAG) [7], which are input to the interpreter. The interpreter is decomposed into two phases: a *virtual machine layer (VML)*, which schedules the DAG and binds its operations to chambers, converting the assay to an intermediate bytecode representation. The *droplet transportation protocol (DTP)* converts each T-type instruction into a path from source-to-destination, and moves all droplets along their respective paths one cell at a time.

The DTP converts the intermediate byte code representation of the assay into machine language: the cells that are activated during each cycle can be derived from the transport information, coupled with the state of each chamber (as set by prior O-type instructions). For example, if a chamber is performing a mixing operation, the cells to activate (at each cycle) are known.

The interpreter can run in either online or offline mode. In offline mode, all scheduling, binding, and routing decisions are made up-front, and the output is a statically compiled sequence of cycles. In online mode, the VML and DTP collaborate to interpret the assay in real-time. The VML schedules and binds assay nodes dynamically, generating O-type instructions (for the operations it wants to perform) and T-type instructions (to transport the droplets to their appropriate destinations before the operations can commence). In real-time, the DTP executes T-type instructions one cell at a time, and informs the VML when each droplet arrives at its destination. The VML executes each O-type instruction when all of the droplets on which the operation depends (as specified by the DAG), arrive at their destinations. For example, when two droplets are set to be mixed, the DTP must route both droplets to the chamber before the VML can execute the O-type instruction that initiates the mixing operation.

4.1 Virtual Machine Layer (VML)

The VML uses *modified list scheduling (MLS)* [7] coupled with a fast and simple binder based on the *left-edge algorithm* [6]. MLS was chosen because of its speed and simplicity. The main goal of the left edge binder is to minimize storage overhead. For example, if two droplets are to be stored, it is better to store them together in one chamber, rather than separately in two chambers. Storing them together maximizes the number of free chambers that become available to perform other assay operations concurrently.

4.2 Droplet Transportation Protocol (DTP)

The primary job of the DTP is to select a path for each droplet and then to route all of the droplets along their respective paths while preventing interference among droplets. If two droplets occupy adjacent cells, they will mix. To prevent this, an *interference region* is defined to be the cells directly adjacent to a droplet (Fig. 3(a)). When a droplet moves, its interference region expands to include the union of the source and destination cells (Fig. 3(b)). As long as no droplet enters the interference region of another, undesired mixing is prevented [1, 3, 5, 9].

The DTP must prevent deadlocks from occurring when multiple droplets are in transport at the same time. To accomplish this goal, we adapted deadlock-free routing algorithms from mesh networks. Our implementation uses a variant of XY routing, but other deadlock-free routing algorithms can also work. Conceptually, XY routing moves each droplet from its source position (x_1, y_1) to its destination position (x_2, y_2), by first traveling along the x-axis to (x_2, y_1) and then traveling along the y-axis to complete the route. XY routing is deterministic and non-adaptive, but worked well enough for our purposes. XY routing prevents specific turns from occurring during the route, as shown in Fig. 3(c).

Several modifications to XY routing are required to account for rotaries (whose internal structure is quite different from mesh routers), the I/O reservoirs on the perimeter of the chip, and the process by which droplets enter and exit chambers. First, we need to introduce some terminology: the four streets and intersections surrounding a chamber form a counter-clockwise traffic circle called a *chamber rotary* (Fig. 4(a)). In Fig. 4(b), *exchange rotaries*, which allow droplets to move from one tile to its neighbors, are formed between tiles (earlier, we referred to them simply as "rotaries"). Larger counter-clockwise cycles can also be formed by combining multiple chamber rotaries (Fig. 4(b)).

We now add four additional rules to the basic XY routing algorithm to prevent deadlock in the DMFB's virtual architecture:

Chamber Entries and Exits: Droplets may not make prohibited turns when leaving source and entering destination chambers. To ensure routability in light of prohibited turns, entries and exits are placed on all four sides of the chamber.

Droplet I/O: To prevent forbidden turns, input, output, and waste reservoirs are placed on the DMFB perimeter and the allowable turns that a droplet may make at an entry point are limited.

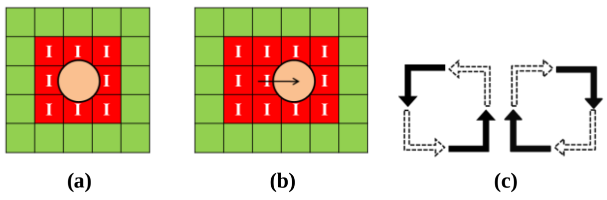

Figure 3. (a) The interference region ('I') of a droplet at the beginning of a cycle; (b) the interference region at the end of a cycle; (c) prohibited turns (white) in XY routing.

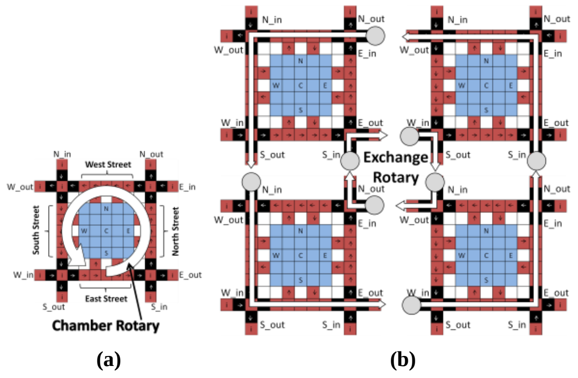

Figure 4. (a) A chamber rotary (the cycle formed by four streets and intersections) (b) An exchange rotary (the clockwise inner loop) and counter-clockwise turns (outer loop) of a tiled design.

Exchange Rotaries: In Fig. 5(a), a droplet *clips* an exchange rotary if it touches one intersection before leaving. In Fig. 5(b) and (c), a droplet *passes through* an exchange rotary if it touches at least two intersections. As droplets move clockwise within an exchange rotary, a clip implies a left turn, and passing through implies that the droplet continues traveling straight or turns right. Fig. 5(d) depicts exchange rotary deadlock when four droplets attempt to pass through; no droplet can progress without violating spacing constraints (Figs. 3(a) and (b)). In Fig. 5(e), deadlock is eliminated if at least one droplet clips the exchange rotary. *To prevent deadlock in an exchange rotary, at most three droplets that wish to pass through may enter concurrently*.

Chamber Rotaries: Fig. 6(a) illustrates chamber rotary deadlock. Droplet 16 creates a dependency chain which causes deadlock; however, if it does not enter the chamber rotary, then a bubble is created which ensures that the sequence of droplets can proceed, starting with Droplet 1. *To prevent deadlock in a chamber rotary, no droplet may enter an exchange rotary unless the system can guarantee that there is space for it to exit into the next street; if the street is full, then the droplet must wait for space to become available prior to entering the exchange rotary.* Droplets attempting to enter a street from an adjacent chamber or input reservoir must also wait until that street has room; in Fig. 6(b), Droplets 1, 2, 3, 5 and 6 must wait for this reason.

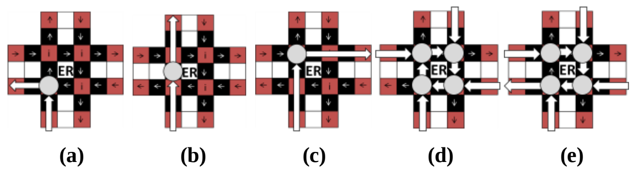

Figure 5. (a) Clipping an exchange rotary ('ER'); (b) Passing through an ER while traveling straight; (c) Passing through an ER while turning right; (d) Deadlocked ER; (e) Non-deadlocked ER.

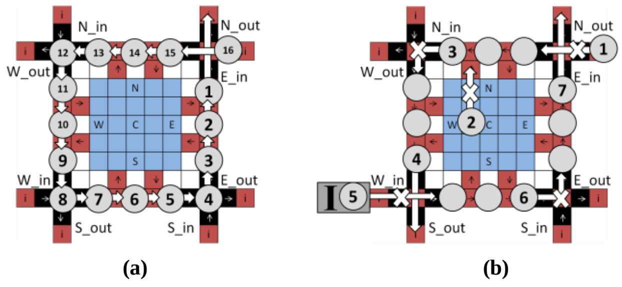

Figure 6. (a) Deadlock in a chamber rotary; (b) Chamber rotary with street capacity rules being enforced to prevent deadlock.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

We implemented our interpreter in C++. We compare against a C++ implementation of a static compilation framework that uses modified list scheduling [7], Su and Chakrabarty's placer [8], and Cho and Pan's high-performance droplet router [3]. We assume a droplet actuation frequency of 100 Hz, as in other works [9].

We evaluated the system on an Inforce SYS9402-01 development board, with a 1GHz Intel Atom™ E638 processor and 512MB RAM, running TimeSys 11 Linux. We chose a low-end processor to show that our interpreter performs well on cheap hardware.

5.2 Experiments Performed

We used the PCR sequencing graph, shown in Fig. 7, for our experiments. Droplet input times are assumed to be 0s. We executed PCR using the two synthesis methods described above. Before obtaining results, we optimized the I/O locations for both methods to ensure that neither method had an unfair advantage. Our online interpreter was run on a 20x20 DMFB (2x2 tile array) and the traditional method on a 17x17 DMFB.

We also ran a stress test on the DTP under a heavy transport load. We generated random traffic on 2x2, 3x3, 4x4, and 8x8 arrays of tiles. For each array, we inject 5 droplets at each input reservoir as quickly as possible. Each droplet stops at two random chambers before exiting at a random output reservoir. These tests required the DTP to route up to 160 droplets concurrently.

5.3 Results and Discussion

Table 1 compares the online interpreter (ON) with the offline compiler (OFF). A routing sub-problem is defined as the time when at least one droplet is routed between assay operations. ON successfully routed the PCR assay in 14ms, while OFF required 10.66 seconds, 769 times longer. The lengths of the computed routes were 670ms and 520ms, respectively; when accounting for scheduling, the total assay completion times were 19.67s and 19.52s respectively. Thus, the total slowdown experienced by our online interpreter was less than 1% of the total execution time. Furthermore, the maximum time ON spent computing routes during any cycle was 2.33ms, well within the 10ms droplet-actuation cycle of a 100Hz DMFB.

Table 2 shows the results of the random traffic stress test. For the largest test, the DTP required 2.7s to compute 480 routes (3 per droplet). The offline method took 10.6s to compute 15 routes for a smaller example. Clearly, the offline router would be a poor choice for use as an interpreter, despite the fact that it produces higher quality routes than the DTP.

6. CONCLUSION

This paper introduced an online assay interpreter for DMFBs and compared it with a static compiler. The router (DTP) can compute routes within milliseconds on low-powered commodity hardware, and the overall impact on assay completion time is negligible. The virtual architecture greatly simplifies placement and routing, but it sacrifices route optimality. A secondary drawback of the virtual architecture is that it is not area efficient, although this is offset by its ability to guarantee deadlock-free routing.

Here, we have established the feasibility of online interpretation. In the future, we plan to exploit the interpreter to introduce new capabilities to the DMFB, such as the ability to execute assays with control flow, to handle variable-latency operations, and to dynamically detect faulty cells and reconfigure the virtual architecture around them. Lastly, we plan to formally verify the correctness of our criteria for deadlock free routing, as proofs were omitted from this paper due to space limitations.

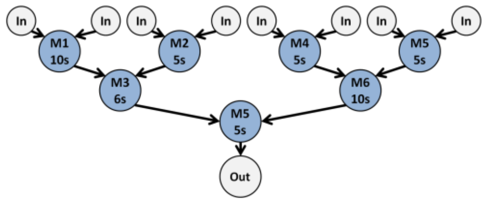

Figure 7. PCR sequencing graph annotated with mixing times.

Table 1. PCR droplet routing results for online vs. offline router.

PCR Routing Results						
Routing Sub-Problem	# Routes		Route Comp. (ms)		Sub-Prob Length (# cycles)	
	ON	OFF	ON	OFF	ON	OFF
1	8	8	5	5294	7	13
2	3	2	6	1626	30	17
3	1	2	2	1620	12	12
4	1	2	1	1469	12	6
5	1	1	0	655	6	4
SUMS:	14	15	14	10664	67	52

Table 2. Random traffic results for our online droplet router.

Random Traffic Stress Test - Online Routing			
DMFB Size (# Chambers)	# Droplets/ #Routes	Completion Time (s)	Total Computation Time (ms)
2x2	40/120	2.19	235.77
3x3	60/180	2.33	514.88
4x4	80/240	2.26	809.57
8x8	160/480	3.69	2780.24

7. ACKNOWLEDGMENTS

This work was supported in part by NSF Grant CNS-1035603. Daniel Grissom was supported by an NSF Graduate Research Fellowship.

8. REFERENCES

[1] K. F. Böhringer. Modeling and controlling parallel tasks in droplet-based microfluidic systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(2): 334-344, February, 2006.

[2] K. Chakrabarty, P. Pop, and T-Y. Ho. Digital microfluidic biochips: recent research and emerging challenges. In Proc. 9th International Conference on Hardware-Software Codesign and System Synthesis (CODES-ISSS), pp. 335-344, Taipei, Taiwan, October 9-14, 2011.

[3] M. Cho and D. Z. Pan. A high-performance droplet router for digital microfluidic biochips. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(10): 1714-1724 November, (2008).

[4] W. J. Dally and B. P. Towles. Principles and practices of interconnection networks. Morgan Kaufammn, 2004.

[5] E. J. Griffith, S. Akella, and M. K. Goldberg. Performance characterization of a reconfigurable planar-array digital microfluidic system. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(2): 345-357, February, 2006.

[6] F. J. Kurdahi and A. C. Parker. REAL: a program for Register Allocation. In Proc. ACM/IEEE Design Automation Conference (DAC), pp. 210-215, Miami, FL, USA, June 28-July 1, 1987.

[7] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. ACM Journal on Emerging Technologies in Computing Systems, 3(4): article #16, January, 2008.

[8] F. Su and K. Chakrabarty. Module placement for fault-tolerant microfluidics-based biochips. ACM Transactions on Design Automation of Electronic Systems, 11(3):682-710, July, 2006.

[9] P-H. Yuh, C-L. Yang, and Y-W. Chang. BioRoute: a network-flow-based routing algorithm for the synthesis of digital microfluidic biochips. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(11): 1928-1941, November, 2008.