Path Scheduling on Digital Microfluidic Biochips

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

ABSTRACT
Since the inception of digital microfluidics, the synthesis problems of scheduling, placement and routing have been performed offline (before runtime) due to their algorithmic complexity. However, with the increasing maturity of digital microfluidic research, online synthesis is becoming a realistic possibility that can bring new benefits in the areas of dynamic scheduling, control-flow, fault-tolerance and live-feedback. This paper contributes to the digital microfluidic synthesis process by introducing a fast, novel path-based scheduling algorithm that produces better schedules than list scheduler for assays with high fan-out; *path scheduler* computes schedules in milliseconds, making it suitable for both offline and online synthesis.

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), Laboratory-on-Chip (LoC), Electrowetting-on-Dielectric (EWoD), Scheduling.

1. INTRODUCTION
This work presents a scheduling heuristic for digital microfluidic synthesis called *Path-scheduler*. Instead of scheduling each node individually, Path-scheduler schedules sets of connected, dependent operations, called *paths,* to increase utilization and yield better schedules for assays with high fan-out. Path-scheduler computes schedules in milliseconds, making it useful for both offline and online scheduling.

1.1 Background
Microfluidics is a laboratory-on-chip (LoC) technology that manipulates fluids on the micro-liter to nano-liter scale to perform biochemical reactions called assays. In contrast to the first generation of microfluidic devices that transport continuous volumes of fluid through channels by actuating pumps and valves, digital microfluidic biochips (DMFBs) manipulate discrete droplets of fluid to perform assays.

A DMFB is arranged as a 2-dimensional array of electrodes, as seen in **Figure 1(a)**. **Figure 1(b)** shows a droplet sandwiched between ground and control electrodes. Although the droplet is centered over CE2, it overlaps neighboring electrodes CE1 and CE3. An activation of CE1 or CE3 will invoke a phenomenon called electrowetting and cause the droplet to flow left or right, respectively, toward the newly-activated electrode [4].

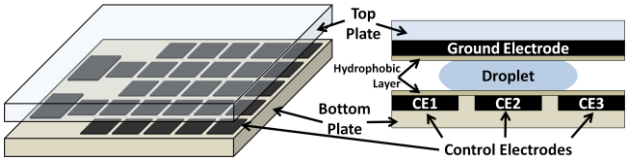


Figure 1. (a) A DMFB 2D array of electrodes; (b) Cross-sectional view of electrode array.

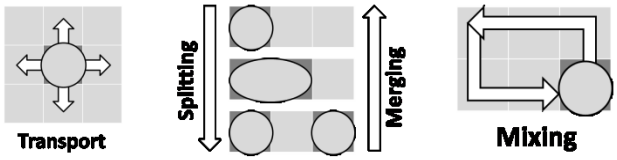


Figure 2. Basic microfluidic operations being performed on 2D array of electrodes.

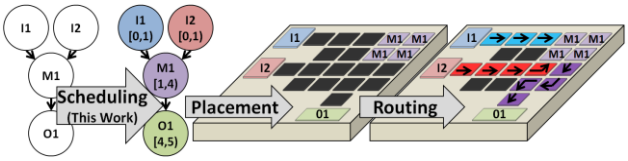


Figure 3. A microfluidic compiler obtains a sequence of electrode activations by *scheduling* a DAG, *placing* DAG operations on the array, and *routing* droplets between operations.

Figure 2 shows how several fundamental microfluidic operations can be performed by activating/deactivating adjacent electrodes in a particular sequence. In addition to droplet transport, splitting, merging and mixing, droplets can be stored on an electrode and input/output from/to reservoirs. Furthermore, individual cells can be equipped with various sensors, cameras and heating elements to perform detection and heating operations [6][11]. These basic operations have been shown adequate to perform an assortment of assays such as in-vitro diagnostics and immunoassays used in clinical pathology [9], DNA polymerase chain reaction (PCR) mixing stages used to amplify DNA [3] and protein crystallizations [12].

A digital microfluidic system consists of two parts: a "wet" array of electrodes, as seen in **Figure 1(a)**, and a "dry" computing device, such as a processor or microcontroller, which sends signals to the microfluidic array to activate electrodes in a pre-determined sequence. This sequence of electrode activations, in turn, causes droplets to perform all the necessary operations (e.g. mixing, merging, transport) to execute an assay.

To obtain the proper sequence of electrode activations, a compiler solves three NP-complete synthesis problems, as seen in **Figure 3**.

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

As input to the compiler, an assay is given as a directed acyclic graph (DAG), which contains the operation dependencies, types and lengths. The compiler first performs resource-constrained scheduling to assign each operation a starting and stopping time-step, ensuring that there are sufficient resources to perform the operations at the scheduled times [3][5][9]. With the newly-scheduled DAG, the compiler then attempts to place/assign each operation to a set of adjacent electrodes at the specified time-steps. Finally, after the operations are placed, droplet routes are computed to transport droplets between dependent operations [2].

1.2 Motivation and Contribution

This paper contributes to the synthesis process by introducing a fast, novel, path-based scheduling algorithm. To date, a list scheduling variant is the fastest scheduling algorithm used for DMFB synthesis [7][9]. When compared to list scheduling, Path-scheduler produces better schedules in competitive times (milliseconds) for assays with high fan-out.

Path-scheduler is also suitable for online synthesis. Currently, assay compilation is performed completely offline due to the complexity of the synthesis process, requiring an assay to be fully-specified before runtime. Ho, Chakrabarty and Pop suggest that "specialized heuristics for the synthesis problems" (scheduling, placement and routing) might enable online synthesis, which would bring new features to DMFBs in the areas of dynamic scheduling, control-flow, fault-tolerance and live-feedback [2]. To date, modified list scheduling (MLS) is the only other scheduler able to generate schedules quick enough to be used in an online manner [7][9]; Path-scheduler can compute better schedules in an online setting for high fan-out assays.

2. RELATED WORK

Our work is closely related to four prior scheduling algorithms that have been proposed in the literature: MLS [7][9], as mentioned above; an optimal scheduler based on integer linear programming (ILP) [9], and two genetic algorithms [5][9]. All of these algorithms solve the resource-constrained scheduling problem, where the DMFB size is known a-priori. This size limits the number of concurrent mixing and storage operations the DMFB can support. The scheduler computes valid start and stop times for each assay operation while ensuring that each DMFB resource is used to process, at most, one operation at each point in time; the objective is to minimize the assay completion time.

Among these four approaches, the genetic algorithms and ILP formulation achieve high quality solutions but with very long runtimes. The genetic algorithms use the iterative improvement paradigm to randomly explore the search space, eventually converging at a local optima; the ILP model employs a commercial solver with an exponential worst-case time complexity to find an optimal solution. Neither of these solutions is appropriate for usage in an online context.

Luo and Akella [3] analyzed a pipelined variant of the PCR assay (**Figure 4(b)**), and developed optimal algorithms for scheduling it (and assays with a similar topology called a "full binary tree") under various resource constraints. In our experiments, both MLS and Path-scheduler found optimal schedules for PCR.

For mixing and dilution operations, larger modules achieve faster runtimes, but consume more area, reducing the availability of resources for other concurrent operations. Su and Chakrabarty [10] developed a genetic algorithm that performs scheduling, module selection, and placement concurrently. The runtime of this algorithm is too large for use in an online context; however, it illustrates the importance of module selection during synthesis. To enable a fair comparison with MLS, we assume uniform module sizes; incorporating module selection into the path scheduling mechanism, without significantly increasing the runtime, is left open for future work.

3. PATH SCHEDULER

3.1 Definitions and Resource Constraints

Let $G = (V, E)$ be a directed acyclic graph (DAG) in which the vertices (V) are the operations of an assay and the edges (E) describe the dependencies between operations. N_m is the total number of *work modules* (the areas where operations are performed) that can be accommodated on the DMFB. A *general module* is a work module that can perform mixing, merging, splitting, and storage operations. A *special-purpose module* is a general module equipped with a sensor, heater, or other external device; special-purpose modules can use these devices to perform additional operations, such as heating and detection. Let N_{si} be the number of special-purpose modules of type i and N_g be the number of general modules; then:

$$\sum_{\forall i} N_{si} + N_g = N_m \quad (1)$$

Any time that a work module is not performing an operation, it is free to be used for storage and can be used to store a maximum of s_c droplets at any given time-step. Let sp_{it} be the number of special-purpose operations of type i being processed at time-step t. Let g_t be the number of general operations being processed and s_t be the number of droplets being stored, respectively, at time-step t. Then, the following inequalities must hold:

$$\sum_{\forall i} sp_{it} + g_t + \left\lceil \frac{s_t}{s_c} \right\rceil \leq N_m, \ \forall t \quad (2)$$

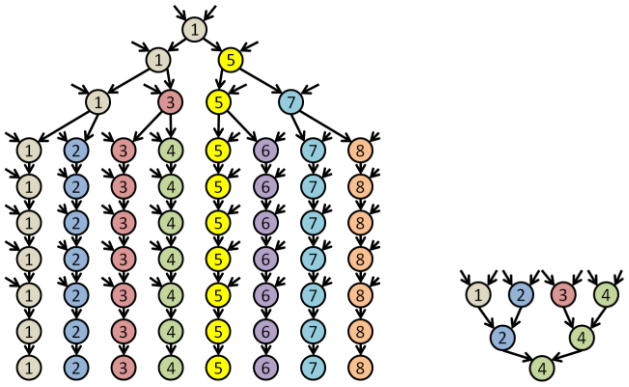
$$sp_{it} \leq N_i \ , \forall i, \forall t \quad (3)$$

In prior work, the feasibility of a given schedule cannot be determined until it is given to a placer and router and successfully mapped to a DMFB. In previous works, a value known as N_a is determined as the number of cells on the DMFB; N_a is then normalized to the number of mixers of a particular size that can fit on the DMFB [9]. N_m, used in this work, is similar to N_a in that it represents the number of work areas of a particular size that have been pre-determined to fit on the DMFB. It is assumed that the modules/mixers can be placed in such a way to allow sufficient room for routing.

Lastly, an I/O port can only process one dispense or output operation at a given time-step and a dispense operation must be bound to a reservoir that contains the appropriate fluid type.

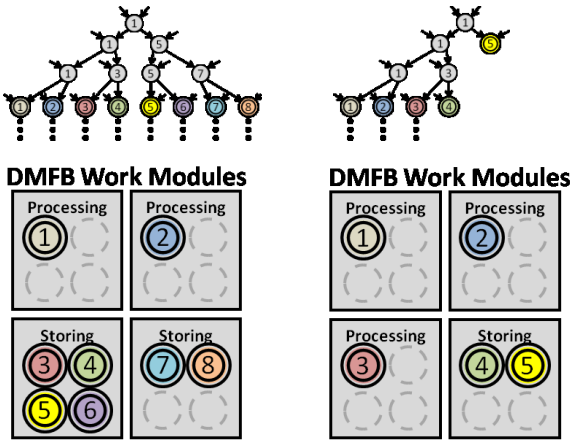
3.2 Approach

A DMFB array contains a fixed number of electrodes on which assay operations can be executed. Unlike a traditional computer where data can be offloaded to a higher-level memory until needed, droplets that are waiting on other dependencies must be stored on the array using the same electrodes that might otherwise be used to perform operations. Consequently, the number of droplets being stored on a DMFB is inversely proportional to the amount of useful work that can be done. Thus, it is important to schedule operations with a goal to minimize the amount of time and number of droplets being stored.



(a) (b)

Figure 4. (a) A protein assay with 8 paths; (b) A PCR mixing assay with 4 paths. Input arrows not attached to a node on both sides represent a dispense operation (nodes omitted for clarity).

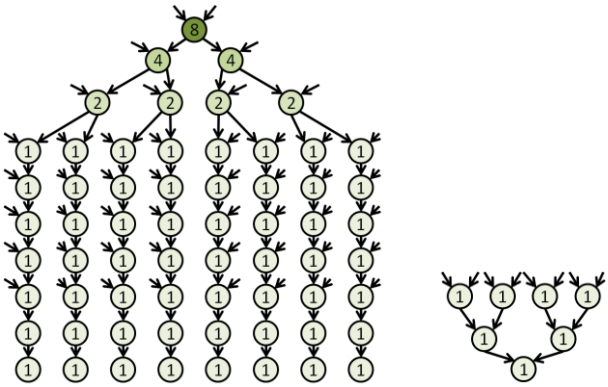


(a) (b)

Figure 5. (a) A non-path scheduler may attempt to process all paths simultaneously, forcing two modules to be used as storage; (b) Path scheduler first schedules paths 1, 2 and 3, and uses only one module to store droplets.

As a simple example, consider the protein assay seen in **Figure 4(a)**, typically used as a benchmark for DMFBs [8][9]. Eight paths can be identified in this assay; path 1 starts with two dispenses, while paths 2-7 originate from one dispense node and one split operation from another path. All paths in **Figure 4(a)** end with an output operation, although this is not the general case, as seen in assays with mix/merge operations (**Figure 4(b)**). It is important to note that the order of paths here is not necessarily unique. Path 1 could be chosen to go down the right side, instead of the left, if all paths from the root split to each output node are the same length. Once the paths are identified, the order in which they are processed is important, as shown in the next paragraph and in Section 3.3.

As seen in **Figure 5(a)**, a scheduler that attempts to schedule single operations at-a-time may attempt to schedule along all eight paths simultaneously (e.g. list-scheduling with a priority function favoring nodes with longer paths to an output) such that there are eight droplets in the system that need to be processed. Assuming the DMFB has enough room for four work modules that can be used to process one operation or store up to four droplets (i.e. $N_m = s_c = 4$), this schedule forces two work modules to be used for storage. However, if operations are first scheduled along paths 1, 2 and 3, only one module is required for storage (see **Figure 5(b)**). This approach effectively prevents droplets from being split until at least one of a split operation's children can be processed, reducing the number of droplets being stored in the system and, in turn, increasing work module utilization.



(a) (b)

Figure 6. The independent-path priorities for (a) a protein assay with high fan-out and (b) a PCR assay with high fan-in.

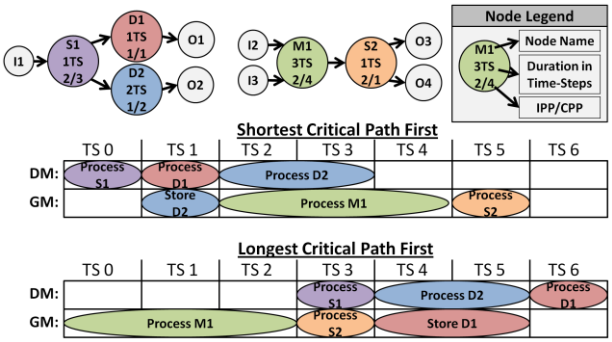


Figure 7. Two schedules for a set of two simple DAGs. The DMFB, in this case, has one general module (GM) and one detect module (DM). Path leaders S1 and M1 both have equal IPPs; if CPP is used as a second, tie-breaking priority, overall runtimes can be reduced if paths with smaller CPPs are chosen first.

3.3 Priority Function

Path scheduler uses two priorities to efficiently produce schedules that minimize the amount of time a droplet spends in the system. To keep droplets from entering the system (via a dispense or split) till as late as possible, path scheduler sets the first priority of each node to the number of *independent paths*, which is the cumulative number of droplets being output in a node's fan-out.

Figure 6 shows the *independent-path priorities* (IPPs) for assays with high fan-out and fan-in (the protein/PCR assays from **Figure 4**). Recall **Figure 5** with the paths and IPPs from **Figure 4(a)** and **Figure 6(a)**, respectively, in mind. The first node on each path is known as the *path leader*. If the scheduler first schedules path 1, leaders from paths 2, 3 and 5 become candidates for scheduling. According to the IPPs in **Figure 6(a)**, path leaders 2, 3 and 5 have priorities of 1, 2 and 4, respectively. By choosing the lowest priority, path 2 is processed next, preventing additional splits from being made until necessary.

Each node has a second priority called the *critical-path priority* (CPP), which is computed as the length of the longest path (in time-steps) from itself to an output node. As seen in **Figure 7**, if the IPP of two candidate nodes (S1 and M1) are the same (2), the path with the lower CPP (S1, CPP = 3) may result in a shorter overall run-time. Since there is only one detect module, one of the

droplets from split-node S1 must be stored in the general module until the other detect finishes. Taking the shortest critical path results in less storage time (1 time-step vs. 2 time-steps in **Figure 7**), which provides more opportunity to increase overall system utilization.

3.4 Algorithm

The pseudocode for the path scheduler algorithm is given in **Figure 8**. Before the scheduling process begins, all nodes in the sequencing graph are assigned first and second priorities as described in Section 3.3 and the candidate operations are determined. The initial candidate operations are those whose parents are only dispense operations.

Lines 6-35 describe the main scheduling process that repeats until all candidate nodes have been scheduled. *Lines 7-9* first select the candidate node, or *path leader*, with the lowest priority value (the lowest IPP first, and in the event of an IPP tie, the lowest CPP), reset the scheduling time-step and initialize an empty path.

Lines 10-24 attempt to allocate resources for an entire path of operations starting with the path leader chosen in *Line 7*, and ending with an output or merge operation (see **Figure 4**). *Lines 11-13* attempt to find the earliest gap in time where the current node, S, can fit, given the available resources. If a gap is found, it means there is a resource of type k (general or special-purpose) available from time-step t_i to $t_i + S.duration$, any required input reservoirs are available and that there are sufficient resources to store any incoming droplets from S's parent nodes, if necessary. If a gap is not found in *Lines 11-13*, it means the current path cannot be scheduled at the moment because of some resource conflict (e.g. there is not enough room to store a droplet from one of S's parents (in path P) to S) ; the path is discarded and any resources being temporarily reserved to schedule path P are relinquished. In this case, Path-scheduler will try to schedule the path again later, but will first return to *Line 6* and attempt to schedule another path.

In the event that a gap is found for S, the starting time-step and resource-type are temporarily saved and S is added to the current path P (*Lines 14-16*); however, S is not marked as scheduled. Once S has been added to path P, *Lines 21-22* select the next node to consider adding to the path from S's children. If the new S is an output or unscheduled mixing operation, then path P is a complete path, is ready to be officially scheduled and can break from the path-constructing loop of *Lines 10-24*; otherwise the loop continues and path scheduler attempts to find a gap for the new S.

Finally, in *Lines 26-34*, each operation in the schedulable path P is marked as scheduled and the resources temporarily reserved in *Line 15* to schedule each of path P's operations are officially reserved in *Line 29*. Also, any unscheduled children of the nodes in path P are added to the candidate operations as path leaders.

The edge between the path node and the new candidate operation added in *Line 31* represents a droplet that must be stored indefinitely and accounted for to properly determine resource availability since it has been scheduled to be created, but may not be used for awhile. Thus, when an operation is added to the candidate list in *Line 31*, the corresponding edge is added to a list of droplets being indefinitely stored from their parent's scheduled ending time-step. When path scheduler is finding a gap for operations in *Lines 11-13*, it considers all droplets being stored indefinitely at that point. When an operation is finally scheduled in *Lines 28-29*, any edges/droplets connected to that node that are being indefinitely stored are removed from the indefinite storage list and the finite period that the droplet must be stored for, if any, is accounted for in the system's available resources.

```
1 Given sequencing graph G=(V, E)
2 Given resource constraints N_gm N_dm and s_c
3 Assign priorities for all nodes v ∈ V, ∀v based on IPP and CPP
4 Find candidate operations R = {v_i ∈ V: Type(v_j) == input, ∀j: (v_j, v_i) ∈ E}
5
6 Repeat {
7    Select S ⊆ R : Priority(S) ≤ Priority(r), ∀r : r ∈ R
8    Time-step t = 1
9    Path P = ∅
10   Repeat {
11      Attempt to find earliest time-step t_i and module-type k for S:
12       k ∈ (gc, sp): Avail(ts, k) == true, ∀ts : t_i ≤ ts < t_i + S.duratoin
13       while holding Equations (2) and (3)
14      if (Attempt found a gap for S)
15         Set S.start = t_i,  Set S.resType = k
16         Add S to P
17      else // Gap could not be found, S not schedulable now
18         Set P.scheduable = false
19      end if
20
21      Select new
22       S ⊆ S.children: Priority(S) ≤ Priority(S_ch), ∀S_ch: S_ch ∈ S.children
23   } until ( ((Type(S) == mix) AND (S.scheduled == false))
24            OR (Type(S) == output ) OR  (P.scheduble == false)   )
25
26   if (P.scheduable == true)
27      for ( ∀p ∈ P ){
28         Set p.scheduled = true
29         Reserve resources for path P
30         for (∀c : c ∈ p.children ∧ c ∉ P )
31            Add c to candidate operations R
32         end for
33      end for
34   end if
35 } until (all candidate operations are scheduled: R = ∅)
```

Figure 8. Pseudocode for the Path-scheduler algorithm.

4. EXPERIMENTAL RESULTS

We implemented our algorithm, as well as two versions of modified-list scheduling in C++; all tests were run on a 64-bit Windows 7 machine with 4GB of RAM, and an Intel Core i7 CPU operating at 2.8GHz.

4.1 Implementation

Our Path-scheduler *(PS)* was implemented as described in **Figure 8**. We re-implemented modified-list scheduling as described in ref. [7] as faithfully as possible. In their work, Su and Chakrabarty describe an *urgency* priority function which sets each node's priority to "the weight of their longest path to the sink" and then "sort[s] them in decreasing order" such that nodes with longer paths are addressed first. We interpret "weight of the longest path" to be calculated in the number of time-steps and compute it similarly to the CPP, described in Section 3.3. Our re-implementation of modified-list scheduling (*MLS_DEC*) processes nodes with higher priorities first.

We also implemented another version of modified-list scheduling with a better-performing priority function. By setting the priorities to our CPP and then sorting them in decreasing order, it causes the scheduler to process a DAG similar to what is seen in **Figure 5(a)**, which is inefficient in terms of utilization. Thus, to be fair to list scheduling, we implemented a new version of modified-list scheduling (*MLS_INC*) which uses CPP for operation priorities, but sorts the nodes in increasing order so nodes with lower priorities are processed first.

4.2 Benchmarks

We used a set of three standard benchmarks: PCR, in-vitro and a protein assay [8]. **Figure 4** displays the protein and PCR DAGs.

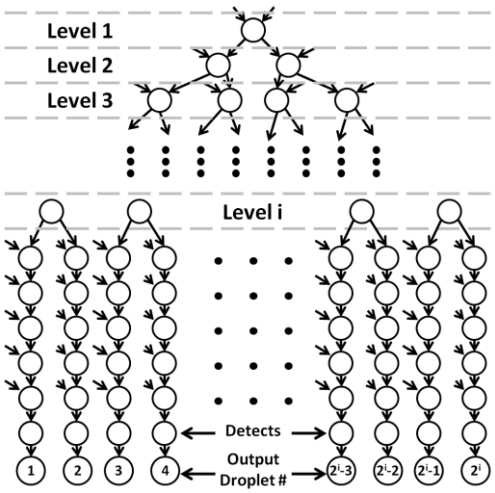


Figure 9. DAG for Experiment 2; *i* levels results in 2^i output droplets.

4.2.1 Assay Annotations

Each benchmark was converted to a DAG and fed to the three schedulers. The module libraries from ref. [8] were used for operation timings. For the PCR assay, a 2x4 mixer (3s) was used for all mixes. The protein DAG was annotated to use the 2x4 diluter (5s) and 2x4 mixer (3s) for all dilute and mix operations, respectively. The 2-input, 2-output dilute operations used in the protein assay were implemented with consecutive mix and split operation which took a cumulative time of 5s. For the in-vitro benchmark, we used the largest sequencing graph which assays four samples with four reagents for a total of 16 mixes/detects. We used the same mix and detect times as detailed in Table 1, Example 5 of ref. [7].

4.2.2 Experiments

For the first set of experiments, we used the PCR, protein and in-vitro DAGs described in Section 4.2.1 as a base. Then, for each of the three DAGs, we attempted to simultaneously schedule an increasing number of copies of the same DAG, from 1 to 10, to show how each scheduler performs with increasing workloads. As a second experiment, we executed 3 protein DAGs while varying the number of modules (N_m) from 2 to 7. In a third set of experiments, we varied the number of splits performed by the protein assay, which allowed us to evaluate the quality of schedules produced by Path-scheduler as assay fan-out increases. In these experiments, we use the protein assay as the source assay. As seen in **Figure 4**, any path from the root split to an output goes through 3 splits before the final string of 7 operations, resulting in 2^3=8 output droplets. Thus, we say that the original protein assay has 3 levels of splits. We sweep the number of split levels from 1-8, so that the number of output droplets sweeps from 2^1=2 to 2^8=256. As seen in **Figure 9**, the final seven operations are appended to the end of each path after the last level of splits.

4.3 Resource Constraints

For the first and third experiment, we set the number of work modules to four (i.e., $N_m = 4$); the second experiment varies N_m. For PCR, all four modules are general modules. For the in-vitro and protein benchmarks, all four modules are detect modules. Similar to ref. [5], we assume a detect module can be used for any detect operation. The PCR and in-vitro benchmarks have one input for each type of fluid used, while the protein benchmark uses one input for sample fluids, two inputs for buffer fluids and two inputs for reagent fluids.

Table 1. Results of Experiment 1. Lower is better for all metrics.

Protein Assay (118 operations/DAG)									
# DAGS	# Time-steps (1TS = 1s)			Scheduling Time (ms)			Avg # Storage Modules/TS		
	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS
1	216	197	186	4	2	1	1.35	0.79	0.64
2	599	342	333	12	5	2	2.50	0.80	0.71
3	990	498	480	20	8	4	2.67	0.83	0.74
4	Fail	643	627	Fail	12	5	Fail	0.83	0.76
5	Fail	799	774	Fail	16	8	Fail	0.85	0.77
6	Fail	944	921	Fail	20	13	Fail	0.84	0.77
7	Fail	1100	1068	Fail	25	14	Fail	0.85	0.78
8	Fail	1245	1215	Fail	30	18	Fail	0.85	0.78
9	Fail	1401	1362	Fail	40	22	Fail	0.85	0.79
10	Fail	1546	1509	Fail	50	29	Fail	0.85	0.79

PCR Assay (16 opearations/DAG)									
# DAGS	# Time-steps (1TS = 1s)			Scheduling Time (ms)			Avg # Storage Modules/TS		
	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS
1	9	9	9	0	0	0	0.00	0.00	0.00
2	15	15	15	0	0	0	0.38	0.19	0.19
3	24	21	21	0	0	0	1.20	0.27	0.27
4	36	27	27	1	1	0	1.46	0.32	0.32
5	45	33	33	2	2	0	1.57	0.35	0.35
6	57	39	39	4	2	0	1.66	0.38	0.38
7	66	45	45	5	3	0	1.70	0.39	0.39
8	78	51	51	8	6	0	1.75	0.40	0.40
9	87	57	57	9	6	0	1.77	0.41	0.41
10	99	63	63	12	8	0	1.80	0.42	0.42

In-vitro Assay (80 operations/DAG)									
# DAGS	# Time-steps (1TS = 1s)			Scheduling Time (ms)			Avg # Storage Modules/TS		
	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS	MLS_DEC	MLS_INC	PS
1	49	44	47	2	1	0	0.68	0.00	0.00
2	119	83	85	9	7	2	1.34	0.00	0.00
3	200	121	123	22	15	3	1.66	0.00	0.00
4	438	159	161	59	27	5	2.58	0.00	0.00
5	580	197	200	101	44	8	2.66	0.00	0.00
6	656	236	238	140	63	10	2.59	0.00	0.00
7	806	273	276	218	88	13	2.66	0.00	0.00
8	915	311	314	265	109	17	2.66	0.00	0.00
9	1065	349	353	364	141	20	2.70	0.00	0.00
10	1174	388	391	442	175	24	2.69	0.00	0.00

4.4 Results and Discussion

Three metrics are presented for evaluation: the completion time in number of time-steps, the time required to compute the schedule and the average number of modules used for storage during each time-step. The last metric represents the scheduler's storage efficiency; the lower this metric, the higher the DMFB utilization.

Table 1 shows the results for Experiment 1. MLS_DEC handles storage poorly, as indicated by its average storage usage; as a consequence, both MLS_INC and PS outperform it significantly. For the protein assay, MLS_DEC attempts to schedule along all paths simultaneously, creating an overwhelming number of storage droplets; for 3 protein DAGs, it already uses 2.67 of 4 modules, on average, for storage. Because of its poor storage handling, it fails to produce feasible schedules for more than 3 protein DAGs because all of the modules are allocated for storage and no assay operations are able to proceed.

The rest of our discussion is limited to MLS_INC and PS. As expected, PS yields great gains with the protein assay, which has high fan-out. In overall schedule quality, PS saves between 9-11 seconds on the first two runs and 37-39 seconds on the last two runs. Furthermore, PS computes schedules 1.94x faster, on average, than MLS_INC. For the interested reader, Supplementary Section S1 shows the scheduled graphs for the protein assay for MLS_DEC, MLS_INC and PS.

Results for the PCR and in-vitro assays are given to demonstrate how PS performs on assays without fan-out since neither of these assays contain a single split. For PCR, PS and MLS_INC yield identical results; PS runs faster than MLS_INC, especially when the number of DAGs increases.

For the in-vitro benchmark, neither PS nor MLS_INC require any storage operations. PS's computed schedules are an average of

Table 2. Results of Experiment 2: scheduling the protein assay using a varying number of modules. Lower is better for all metrics.

Protein Assay - 3 DAGs						
# Mods	# TS (1TS = 1s)		Sched Time (ms)		Avg # Stor Mod/TS	
	MLS_INC	PS	MLS_INC	PS	MLS_INC	PS
2	Fail	1177	Fail	3	Fail	0.90
3	775	590	9	2	1.39	0.86
4	480	407	8	1	1.36	0.78
5	345	305	5	1	1.36	0.70
6	295	255	6	1	1.88	0.66
7	250	217	7	1	1.83	0.74
25	60	60	2	1	0.00	0.00

Table 3. Results of Experiment 3: varying the number of splits in the protein assay. Lower is better for all metrics.

Protein Split Assay						
# Split	#TS (1TS = 1s)		Sched Time (ms)		Avg # Stor Mods/TS	
Levels	MLS_INC	PS	MLS_INC	PS	MLS_INC	PS
1	71	72	0	0	0.28	0.21
2	106	110	0	0	0.49	0.43
3	197	186	2	1	0.79	0.64
4	389	338	5	2	1.07	0.79
5	757	642	13	4	1.30	0.91
6	1590	1279	29	14	1.54	1.03
7	3456	2644	73	24	1.78	1.15
8	Fail	5570	Fail	49	Fail	1.29

2.7s slower than MLS_INC. The root of this small inefficiency is actually due to input-reservoir conflicts. A conflict resolution step was added to PS and it improved the results for in-vitro to be equivalent with MLS_INC; however, due to its inherent quadratic runtime, this step increased the runtime of PS significantly (e.g. from 24ms to 7439ms for in-vitro run #10) and did not yield improvements for protein or PCR schedules. We describe these issues in greater detail in Supplementary Section S2. Lastly, we note that the runtime of PS is 3.5x to 7.3x faster than MLS_INC in this experiment.

Table 2 shows results for Experiment 2, where we varied the number of work modules while scheduling the protein assay. For 2 modules, PS completed a schedule in 1177 time-steps, while MLS_INC failed to compute a schedule. For $N_m = 3$, 4, 5, 6 and 7, PS's schedules are 185, 73, 40, 40 and 33 time-steps shorter than MLS_INC's schedules, respectively. As the number of modules increase, the schedules tend to converge since there are abundant resources for storage and it becomes algorithmically easier to compute latency-optimal schedules.

Table 3 reports results for Experiment 3, which confirms that PS generates better schedules than MLS_INC when fan-out increases by reducing storage usage. PS loses 1-4 time-steps for the first two levels of splits. However, as the number of splits increase from three to seven levels, PS saves up to 812s (13m 32s). Due to poor memory management, MLS_INC fails to produce schedules beyond seven split levels because it reaches a point where all four modules are used for storage. With the constraints detailed in Section 4.3, PS can theoretically compute schedules up to 12 levels (4096 droplets). We verified this experimentally: PS took 1269ms (1.3s) to compute a schedule of 31h 29m in length.

5. CONCLUSION

We have presented a path-based scheduling heuristic for digital microfluidic synthesis. Instead of scheduling node-by-node, as list scheduler does, Path-scheduler schedules path-by-path, reducing the number of droplets being stored in the system on assays with high fan-out. The increase in storage efficiency leads to an increase in utilization, and in turn, an increase in overall schedule quality. Similar to list scheduling, Path-scheduler produces solutions on the order of milliseconds. As assays grow extremely large, as seen in Experiment 3, the schedules generated by Path-scheduler will be further appreciated when the compiler attempts to place and route a much smaller schedule.

As synthesis moves to an online setting, short runtimes become increasingly important and assays will likely be scheduled with specialized scheduling heuristics that perform well on a particular assay-class (e.g. multiplexed, high fan-out/in, etc.). Path-scheduler excels on and should be used on assays with high fan-out (easily obtainable info). Even with no a-priori information, Path-scheduler is fast enough that a DMFB could compute schedules with path- and list scheduler and take the best schedule with little penalty. As more fast, high-quality scheduling heuristics emerge, online synthesis will become a growing possibility, bringing a number of new features to DMFBs in the areas of dynamic scheduling, control-flow, fault-tolerance and live-feedback.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] K. Chakrabarty. Design automation and test solutions for digital microfluidic biochips. IEEE Transactions on Circuits and Systems-I: Regular Papers, 57(1):4-17, January 2010.

[2] T. Ho, K. Chakrabarty, and P. Pop. Digital microfluidic biochips: recent research and emerging challenges. In Proceedings of the Conference on Codesign and System Synthesis, pages 335-343, Taipei, Taiwan, Oct 9-14, 2011.

[3] L. Luo and S. Akella. Optimal scheduling of biochemical analyses on digital microfluidic systems. In Proceedings of the Conference on Intelligent Robots and Systems, pages 3151-3157, San Diego, CA, USA, Oct 29-Nov 2, 2007.

[4] M. G. Pollack, A.D. Shenderov, and R. B. Fair. Electrowetting-based actuation of droplets for integrated microfluidics. Lab on a Chip, 2:96-101,2002.

[5] A. J. Ricketts, K. Irick, N. Vijaykrishnan, and M. J. Irwin. Priority scheduling in digital microfluidics-based biochips. In Proceedings of the Conference on Design Automation and Test in Europe (DATE), pages 329-334, Munich, Germany, March 6-10, 2006.

[6] V. Srinivasan, et al. A digital microfluidic biosensor for multianalyte detection. Proc. IEEE MEMS, pages 327-330, Kyoto, Japan, Jan 19-23, 2003.

[7] F. Su and K. Chakrabarty. Architectural-level synthesis of digital microfluidics-based biochips. In Proceedings of ICCAD, pages 223-228, San Jose, CA, USA, Nov 7-11, 2004.

[8] F. Su and K. Chakrabarty. "Benchmarks" for digital microfluidic biochip design and synthesis. Duke University, Department of Electrical and Computer Engineering, 2006. http://www.ee.duke.edu/~fs/Benchmark.pdf

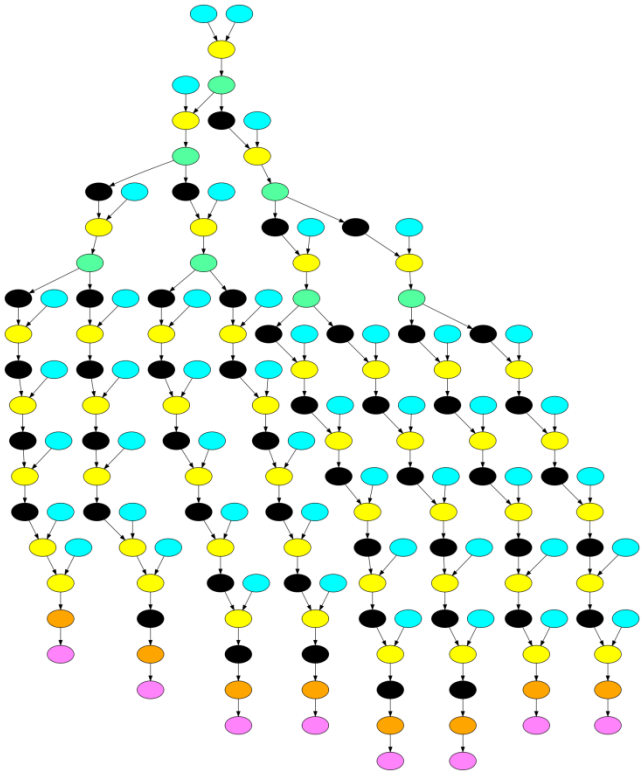
[9] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. ACM Journal on Emerging Technologies in Computing Systems, 3(4): article #16, January, 2008.

[10] F. Su and K. Chakrabarty. Unifed high-level synthesis and module placement for defect-tolerant microfluidic biochips. In Proceedings of Design Automation Conference, pages 825-830, Anaheim, CA, USA, June 13-17, 2005.

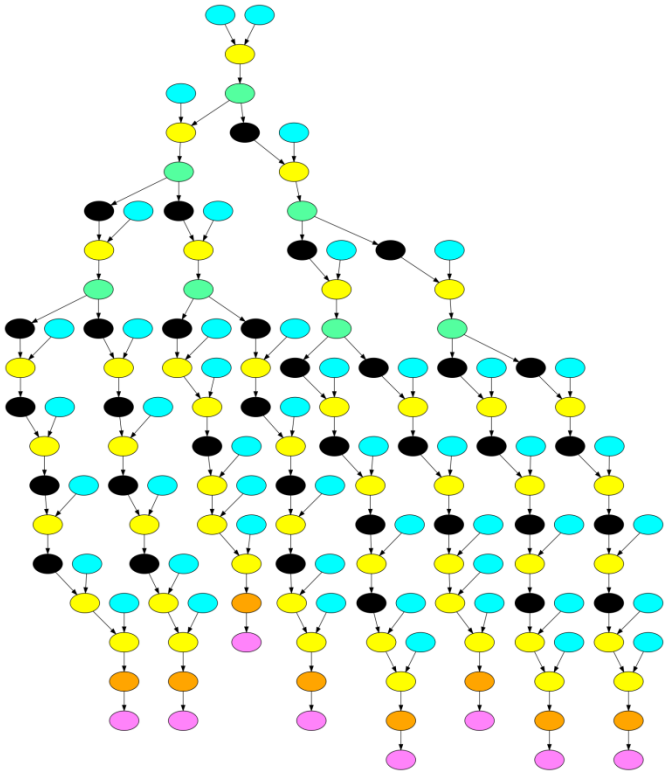
[11] K. Ugsornrat, et al. Experimental study of single-plate EWOD device for a droplet based PCR system. In Proceedings of ECTI-CON, pages 6-9, Khon Kaen, Thailand, July 12, 2011.

[12] T. Xu, K. Chakrabarty, and V. K. Pamula. Defect-tolerant design and optimization of a digital microfluidic biochip for protein crystallization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(4): 552-565, April, 2010.

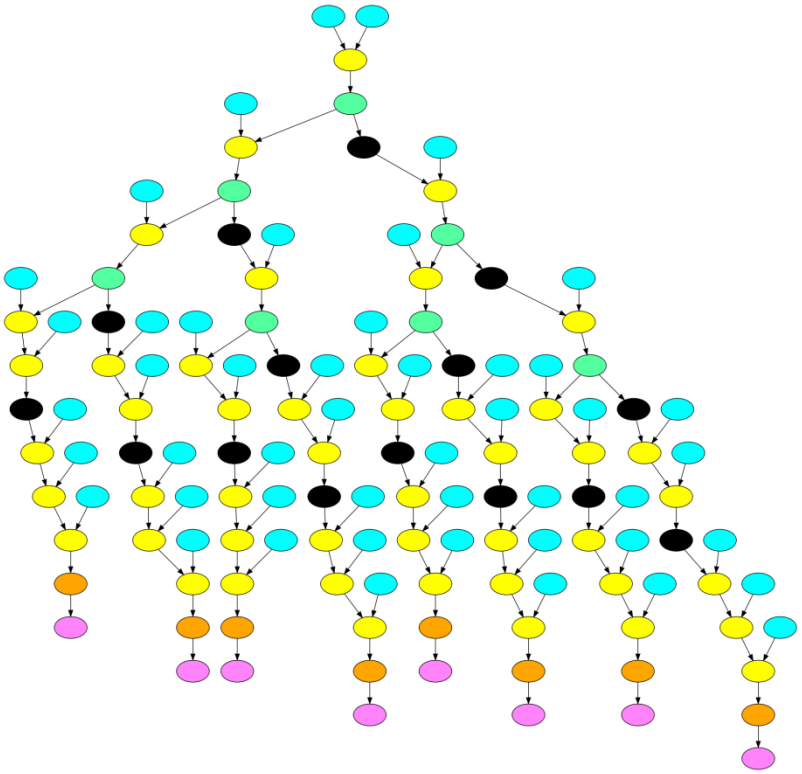
S1. STORAGE REDUCTION GRAPHS



MLS_DEC (highest priorities first) – 48 Storage Nodes



MLS_INC (lowest priorities first) – 34 Storage Nodes



PS (Path Scheduling) – 15 Storage Nodes

Figure S1. Scheduled DAGs for the basic protein assay (timing information was removed for clarity). These graphs have not been bound (placed), and thus, each black storage node represents a droplet being stored for a length of time, possibly in a number of different modules.

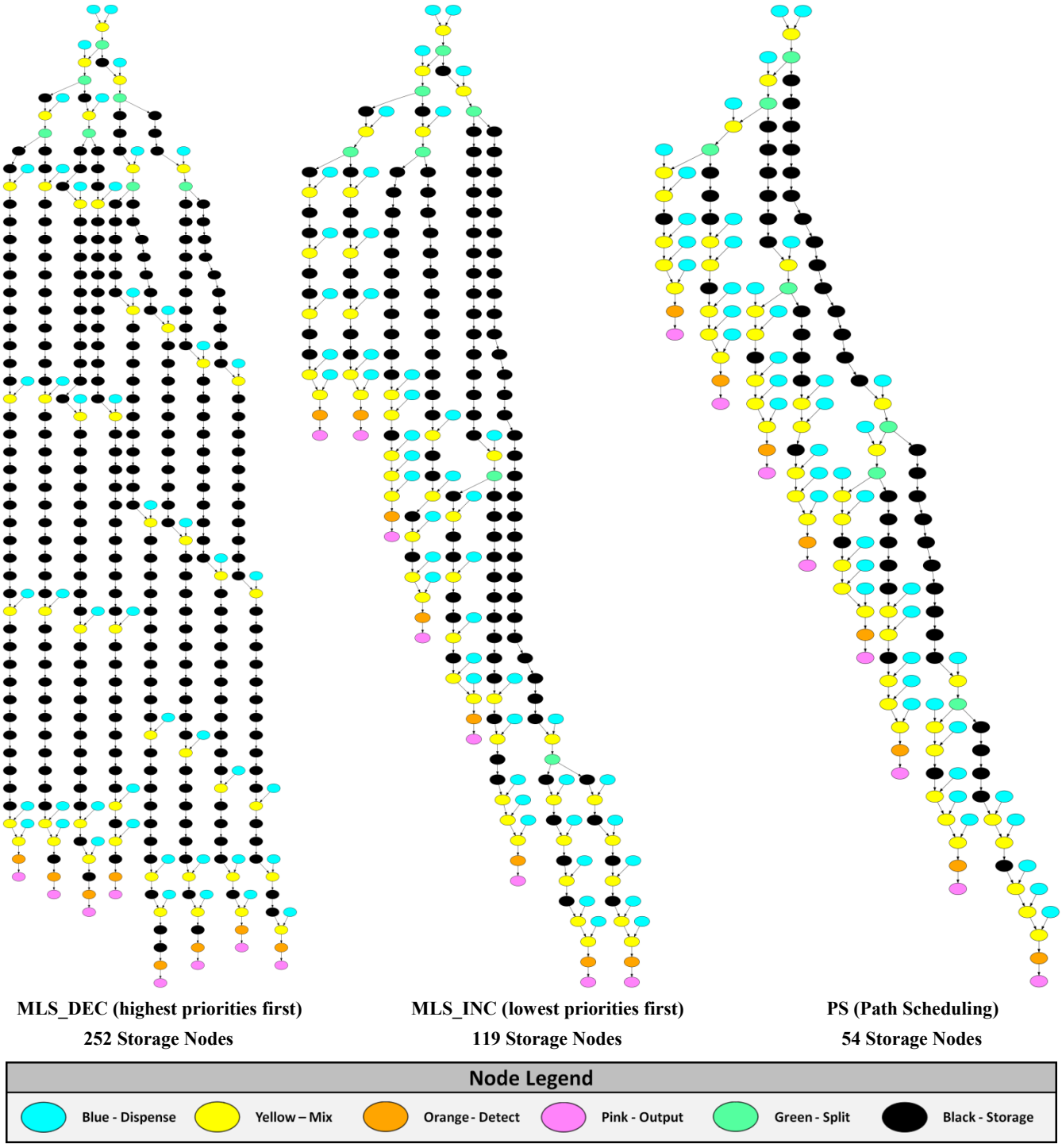


Figure S2. Scheduled and bound DAGs for the basic protein assay. Each black storage node represents a droplet being stored for an amount of time in a particular module.

Figure S1 shows the scheduled graphs for the basic 118-node protein assay for MLS_DEC, MLS_INC, and PS. The main points of interest are the locations and number of storage nodes. Each storage node represents a droplet being stored for a number of time-steps, possibly in a number of different locations. MLS_DEC inserts storage nodes in-between almost every pair of consecutive non-dispense nodes for a total of 48 storage nodes. MLS_INC is slightly more conservative, while PS inserts only 15 storage nodes, usually after splits occur.

Figure S2 shows the same protein assay that is now scheduled and bound to one of the four specific work modules. Here, the black storage nodes from **Figure S1** have been unrolled and represent a droplet being stored for a number of time-steps in a particular work module. Although placement/binding is beyond

Table S1. Mix times according to in-vitro sample-type.

Sample-Type Mix Times			
S1	S2	S3	S4
5s	3s	4s	6s

Table S2. Detect times according to in-vitro reagent-type.

Reagent-Type Detect Times			
R1	R2	R3	R4
5s	4s	6s	5s

Table S3. Experimental data from experiment 1 with results from PS_IN.

In-vitro Assay (80 operations/DAG)												
# DAGS	# Time-steps (1TS = 1s)				Scheduling Time (ms)				Avg. # Storage Modules/TS			
	MLS_DEC	MLS_INC	PS	PS_IN	MLS_DEC	MLS_INC	PS	PS_IN	MLS_DEC	MLS_INC	PS	PS_IN
1	49	44	47	45	2	1	0	7	0.68	0.00	0.00	0.00
2	119	83	85	83	9	7	2	66	1.34	0.00	0.00	0.00
3	200	121	123	122	22	15	3	225	1.66	0.00	0.00	0.00
4	438	159	161	159	59	27	5	550	2.58	0.00	0.00	0.00
5	580	197	200	197	101	44	8	1011	2.66	0.00	0.00	0.00
6	656	236	238	236	140	63	10	1700	2.59	0.00	0.00	0.00
7	806	273	276	273	218	88	13	2594	2.66	0.00	0.00	0.00
8	915	311	314	311	265	109	17	3779	2.66	0.00	0.00	0.00
9	1065	349	353	350	364	141	20	5553	2.70	0.00	0.00	0.00
10	1174	388	391	387	442	175	24	7439	2.69	0.00	0.00	0.00

the scope of this paper, we created **Figure S2** from the scheduled graphs seen in **Figure S1** using a simple left-edge binder to better-demonstrate how the various schedules will execute. These results show that MLS_DEC tries to execute all paths concurrently, resulting in much greater demand for storage than MLS_INC or PS. In contrast, PS processes the left side of the assay first, while storing droplets at a few strategic locations on the right side.

S2. IN-VITRO INEFFICIENCY

In this section, we show that the slightly inferior schedules produced by PS for the in-vitro benchmark arise from input reservoir resource conflicts. In the basic 80-node in-vitro benchmark, four samples (S1-S4) are pairwise assayed with four reagents (R1-R4) yielding a DAG with 16 connected components. Each sample-reagent pair is mixed together, sent to a detector for evaluation, and then output. Mix times (**Table S1**) are based on the sample type, while detect times (**Table S2**) are based on the reagent type. All dispense times are 2s.

Figure S3 shows what choices MLS_INC and PS make for the first four assays. Since the IPPs for PS all equal 1, PS and MLS_INC both use only CPPs to help decide which paths/nodes to schedule next. Both choose mix node M6, which mixes S2 and R2 together because the CPP of 7TS (3TS for M6 and 4TS for D6) is the smallest of any sample-reagent pair. From **Table S1** and **Table S2**, the assays with the next three lowest priorities, all tied at 8TS, are S2-R1 (3s+5s), S2-R4 (3s+5s) and S3-R2 (4s+4s). **Figure S3** reveals that PS's first four scheduled DAGs include these 3 assays. However, notice that S2 and R2 were already scheduled and that each of the next three assays contain either S2 or R2. PS first tries to schedule the input operations for S2-R1 at time-step 0, but finds that there is a resource conflict with the input port dispensing S2. Instead of moving to a new connected component, PS sticks with the current one and schedules it as soon as the input port for S2 is available at time-step 2. PS finds that its next component, which assays S3 with R2 (chosen next, at random, from the 3-way tie) has a resource conflict with R2 and must also start at time-step 2. Finally, PS receives the connected component with S2 and R4 and cannot schedule it until time-step 4 because the input port for S2 is busy during the prior time-steps.

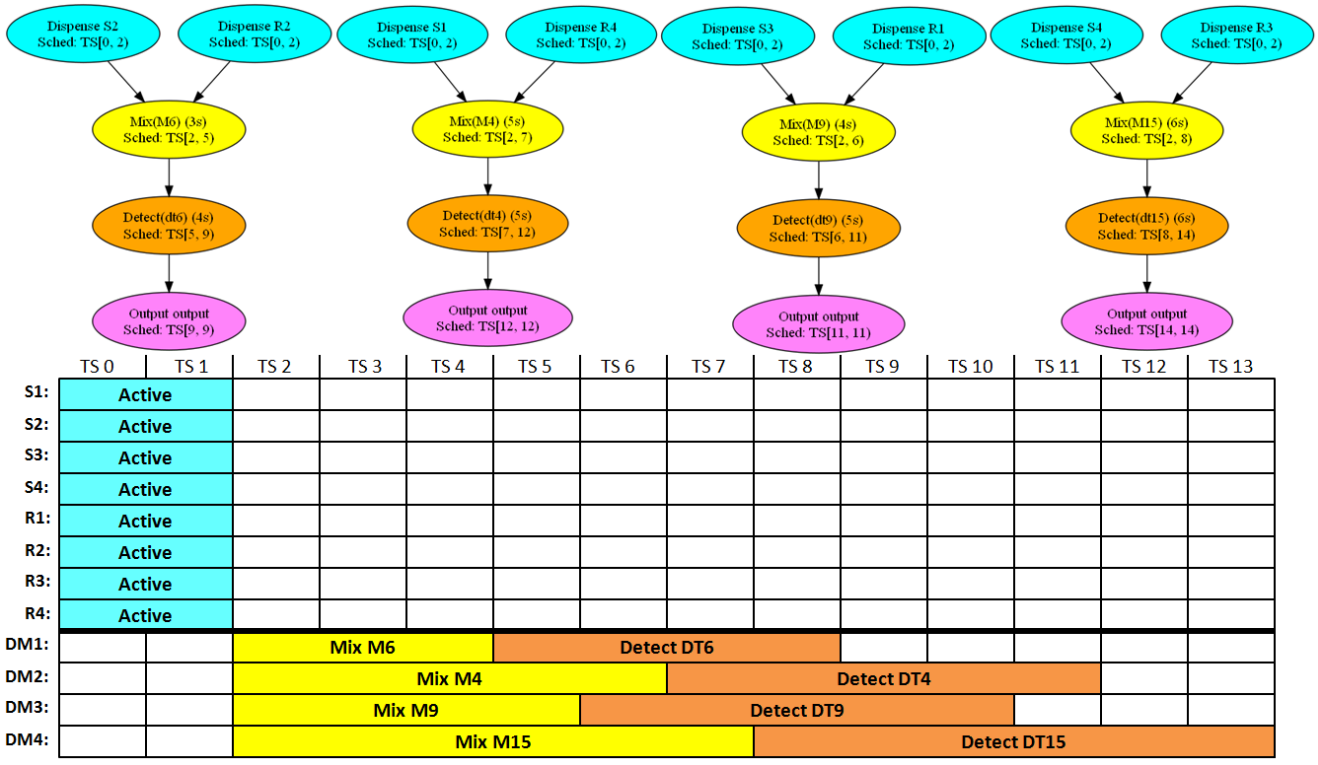
If the random order of the ties is the same in MLS_INC and PS, MLS_INC will examine the assays in the same order. However, the behaviors of PS and MLS_INC cause the schedules to differ at this point. PS can go back and add to a partial schedule that is already in place. For example, if PS has already scheduled a path from time-steps 0-10, it can revisit those time-steps and schedule another path from time-steps 0-10. However, list scheduling (including, but not limited to MLS_INC) takes a constructive approach and does not revisit a time-step once it has moved to the next. For example, if MLS_INC is scheduling a node at time-step 5, it can never go back to time-steps 0-4 to add to the schedule.

When MLS_INC reaches time-step 2, it schedules M6, along with its parent nodes for S2-R2. MLS_INC also examines the three assays with path-lengths (or CPPs) of 8TS, but finds that none can be scheduled at the moment because they all require the input port for S2 or R2. Since MLS_INC cannot come back to this time-step, it greedily examines all assays until it finds three that do not conflict with the inputs of the first assay or with each other. **Figure S3** shows that MLS_INC is able to utilize all input ports in the first two time-steps. Once the first four mixers (and corresponding dispense parents) are scheduled, the DMFB runs out of modules and list scheduler moves to the next time-step.

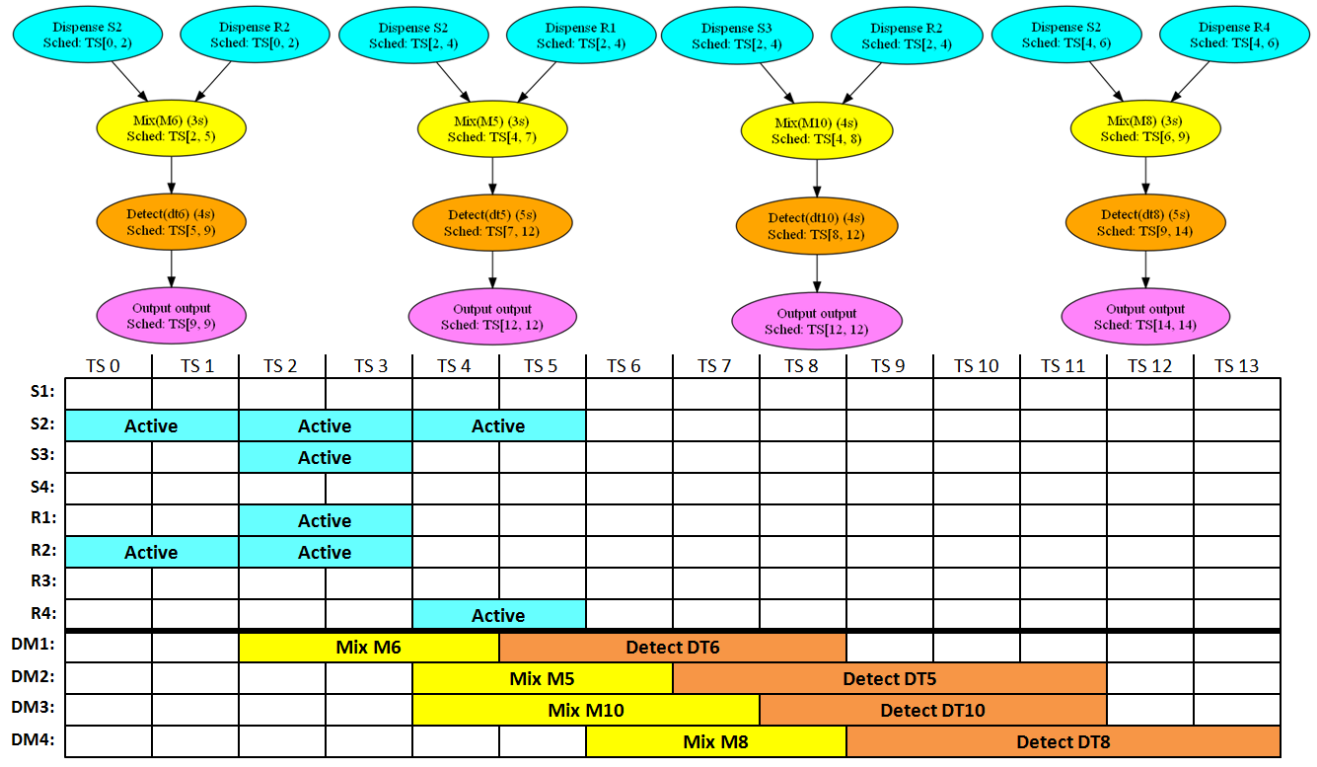
MLS_INC only releases one module (DM3) before PS. However, even though three of the four modules finish their first assay at the same time for both schedules, MLS_INC has scheduled longer assays, and thus, has less work than PS left to schedule. Both schedulers have nearly identical amounts of resources left due to the fact that PS expended more resources at the beginning of its schedule. This slight skew in scheduling is the cause of the 3-4 time-step deficiencies seen in the PS schedules.

Thus far, we have shown that the inefficiency for the multiplexed in-vitro benchmark is due to resource conflicts with the input reservoirs. The solution is to add some priority to the order in which paths are scheduled based on the input reservoirs required by that path. To test this idea, we modified Path-scheduler so that it no longer considered the CPP as a second priority. If there were a number of assays (mix nodes) with the same IPP priority, the modified Path-scheduler (PS_IN) examines all parents of the tied assays/nodes and schedules the mixers whose dispense parents are available earliest. This solution is quadratic in the number of paths, as it searches through the list of unprocessed (tied) nodes each time PS_IN chooses a new path to schedule.

Table S3 shows the results of PS_IN alongside the original results for the in-vitro diagnostic test from experiment 1. The schedules for PS_IN are now within one time-step (+/-) of the MLS_INC schedules. We attribute the one time-step differences to randomness in the order that ties are ordered by each scheduler. As expected, runtimes for PS_IN are much longer than PS or MLS_INC, which arguably outweighs the benefits of improving the schedule quality by 2 to 3 time-steps.



MLS_INC (lowest priorities first)



PS (Path-scheduler)

Figure S3. Shows the first four scheduled mixes, along with bindings to work modules and inputs for MLS_INC and PS. S1-S4, R1-R4 and DM1-DM4 represent the four sample input reservoirs, reagent input reservoirs and detect modules, respectively, on the DMFB.