Online Synthesis for Operation Execution Time Variability on Digital Microfluidic Biochips

Mirela Alistar, Paul Pop
Technical Univ. of Denmark, Lyngby, Demark, email: paupo@dtu.dk

***Abstract*—Several approaches have been proposed for the synthesis of digital microfluidic biochips, which, starting from a biochemical application and a given biochip architecture, determine the allocation, resource binding, scheduling, placement and routing of the operations in the application. Researchers have assumed that each biochemical operation in an application is characterized by a worst-case execution time (*wcet*). However, during the execution of the application, due to variability and randomness in biochemical reactions, operations may finish earlier than their *wcet*s. In this paper we propose an online synthesis strategy that re-synthesizes the application at runtime when operations experience variability in their execution time, obtaining thus shorter application execution times. The proposed strategy has been evaluated using several benchmarks.**

I. Introduction

Microfluidic biochips have the potential to replace the conventional laboratory equipment as they integrate all the functions needed to complete a bioassay. Applications on biochips are considered in areas such as, *in vitro* diagnostics (point-of-care, self-testing), drug discovery (high-throughput screening, hit characterization), biotech (process monitoring), ecology (environment, homeland security) [3].

In this paper, we are interested in *digital microfluidic biochips* (DMBs), which manipulate the fluids as droplets (discrete amount of fluid of nanoliters volume), using electrowetting-on-dielectric (EWOD) [3]. DMBs are able to perform fluidic operations such as dispensing, transport, mixing, split, dilution and detection.

To be executed on a DMB, a biochemical application has to be synthesized [5], [3], [9]. All synthesis strategies proposed so far in related research (the only exception is [7]) consider a given module library $\mathcal{L}$, which contains for each operation its worst-case execution time (*wcet*). However, an operation can finish before its *wcet*, due to variability and randomness in biochemical reactions [6]. Such situations, when the actual execution time of the operation is less than the *wcet*, result in time slacks in the schedule of operations. These time slacks can be used for executing other operations in the application, thus, reducing the application completion time. Besides reduced costs, due to shorter experimental times, reducing the application execution time can also be beneficial for fault-tolerance. For example, researchers have shown [1], [8] how the slack to the application deadline can be used to introduce recovery operations to tolerate transient faults.

In this paper we propose a strategy to exploit the slack time resulted due to uncertainties in operation execution, aiming at minimizing the application completion time.

The only work that addresses variability in operation execution is [7], which proposes an Operation-Interdependency-Aware (OIA) synthesis to derive an offline implementation that is scaled at runtime according to the actual operation execution scenario. The strategy of OIA is to group the operations according to their type and scheduling constraints and then to schedule the operations in phases. The phases are executed alternatively, and each phase is considered completed when all its operations finish executing. Although OIA can handle any variability in the operation execution, its approach, where all operations of the same phase have to wait for each other to finish, is overly pessimistic and leads to long application completion times.

In this paper we first propose an *online* synthesis strategy (called ONS) that, when an operation finishes earlier than its *wcet*, runs a re-synthesis to derive a new solution, aiming at minimizing the application completion time. Because it is executed at runtime, our online synthesis strategy has the advantage of taking into account the actual operation execution times, successfully adapting the binding, placement, routing and scheduling of operations. The disadvantage of an online approach is its overhead due to multiple runtime re-syntheses which add delays to the application completion time. However, an online re-synthesis can be a viable strategy because the execution of the synthesis tasks on the computer is orders of magnitude smaller compared to typical biochemical operation completion times, see Table I.

II. Biochip architecture model

In a DMB, a droplet is sandwiched between a top ground-electrode and bottom control-electrodes. The droplets are manipulated using the EWOD principle [3]. A biochip is typically connected to a computer (or microcontroller) and it is controlled based on an "electrode actuation sequence" that specifies for each time step which electrodes have to be turned on and off, to run a biochemical application.

A DMB is modeled as a two-dimensional array of identical control-electrodes, see Fig. 2, where each electrode can hold a droplet. There are two types of operations: reconfigurable (mixing, split, dilution, merge, transport), which can be executed on any electrode on the biochip, and non-reconfigurable (dispensing, detection), which are bound to a specific device with fixed location, such as a reservoir, a detector or a sensor.

Each reconfigurable operation is executed in a determined biochip area, called a "module" surrounded by a "segregation border" to avoid accidental merging, see Fig. 2.

978-1-4799-4833-8/14/$31.00

TABLE I
MODULE LIBRARY $\mathcal{L}$

Operation	Module area	*bcet* (s)	*wcet* (s)
Dispensing	N/A	1	2
Mix	3×6	2	3.47
Mix	4×6	1.5	2.5
Mix	4×2	2.5	4.3
Mix	4×3	2.3	4
Dilution	3×6	2.3	4
Dilution	4×6	1.5	3.1
Store	1×1	N/A	N/A
Transport	1×1	0.01	0.01

With the exception of [7], all past research has assumed that each operation executes in a given *wcet*. Thus, based on experiments, researchers characterize a module library $\mathcal{L}$, such as the one in Table I, which provides the area and corresponding best-case execution time (*bcet*) and *wcet* for each operation.

Recent work [7], [1] has addresses the cybephysical integration of the biochip and the control system. In such a setup, the biochip is equipped with a "sensing system" [4] that, during the execution of an operation, monitors attributes such as color, concentration, volume, diameter and position [7]. Similar to [7], in this paper, we assume that we are able to also determine the completion time of an operation.

III. BIOCHEMICAL APPLICATION MODEL

A biochemical application is modeled using an acyclic directed graph [3], where the nodes represent the operations, and the edges represent the dependencies between them.

In this paper, we denote with $\mathcal{G}$ the biochemical application model, such as the one in Fig. 1. A node in $\mathcal{G}$ represents an operation O_i, thus in Fig. 1 we have operations O_1 to O_{15}. A directed edge e_{ij} between operations O_i and O_j models a dependency: O_j can start to execute only when it has received the input droplet from O_i. We consider that an application $\mathcal{G}$ is characterized by a deadline $d_{\mathcal{G}}$.

IV. PROBLEM FORMULATION

As an input, we have a biochemical application $\mathcal{G}$, with a deadline $d_{\mathcal{G}}$, to be executed on a given biochip architecture $\mathcal{A}$. A characterized module library $\mathcal{L}$ containing the area, *bcet* and *wcet* for each operation, is also given as input. We are interested to determine an implementation Ψ, which minimizes the application completion time $\delta_{\mathcal{G}}$ in case of uncertainties in operation execution times. Note that the actual operation execution times will only be known during the execution of the application, once an operation completes.

Deriving an implementation Ψ means deciding on the *allocation* $\mathcal{O}$, which selects the devices to be used from the library $\mathcal{L}$, the *binding* $\mathcal{B}$, which decides what operations to execute on the allocated modules, the *placement* $\mathcal{P}$, which decides the positions of the modules on the architecture $\mathcal{A}$, the *schedule* $\mathcal{S}$, which decides the order of operations and the *routing* $\mathcal{U}$, which determines the droplets routes.

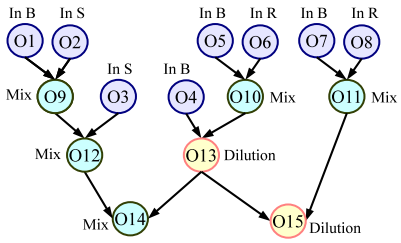

Fig. 1. Example application $\mathcal{G}$

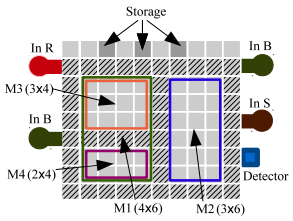

Fig. 2. Placement of modules

A. Motivational example

To illustrate our problem let us use as an example the application graph $\mathcal{G}$ from Fig. 1, which has to execute on the 10×9 biochip $\mathcal{A}$ in Fig. 2. The deadline is $d_{\mathcal{G}} = 13$ *s*. We consider that the operations are executing on rectangular modules which have their area, *bcet* and *wcet* specified in the library $\mathcal{L}$ from Table I. For this example, we assume that the placement of the modules is presented as in Fig. 2. Also, we ignore routing for simplicity reasons in all the examples in the paper, but we implement routing in all our experiments.

Researchers have so far proposed design-time algorithms that use the *wcet*s for the operation execution times. Such a solution is presented in Fig. 3a, and the resulted application completion time $\delta_{\mathcal{G}}$ is 13 *s*. The schedule is depicted as a Gantt chart, where for each module, we represent the operations as rectangles with their length corresponding to the duration of that operation on the module.

Let us assume the execution scenario where operations O_{10-15} finish sooner than their respective *wcet*. The optimal implementation, considering this execution scenario, is presented in Fig. 3b, where we also show, right above the rectangles representing each operation, the actual execution times of O_{10-15}, and in parentheses their corresponding *wcet*. For example, the actual execution time of O_{10} is 2 thus O_{10} finishes at $t = 4$, instead of $t = 4.5$ if O_{10} would have

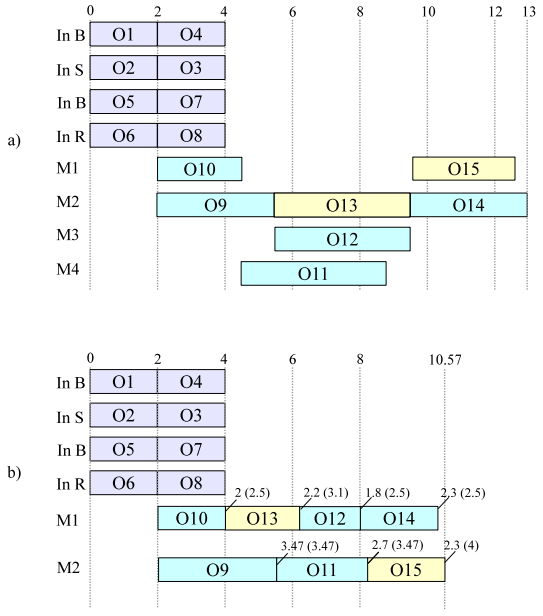

Fig. 3. (a) Offline synthesis (b) Online synthesis

completed in its $wcet = 2.5$. The implementation in Fig. 3b has a new allocation, binding, placement and schedule such that the application completion time is minimized. As we can see, by using the actual execution times for the operations, and not their *wcet*s, we can improve $\delta_{\mathcal{G}}$ from 13 *s* to 10.57 *s* (i.e., an improvement of 18.6%).

The challenge is that we do not know in advance, at design time, which operations will finish earlier and their execution times. The actual operation execution scenario is only known at runtime, as detected by the available sensing system.

The only work that addresses the uncertainties in operation execution problem is [7], where an Operation-Interdependency-Aware (OIA) synthesis is proposed to derive an offline schedule that is scaled at runtime depending on the actual operation execution times. As an alternative to OIA we propose an Online Synthesis method (ONS) which determines at runtime a new implementation Ψ every time we detect that an operation finishes before its *wcet*.

V. Online Synthesis Strategy (ONS)

We propose an Online Synthesis Strategy (ONS) to solve the problem formulated in Section IV. As depicted in Fig. 4, ONS is run at runtime each time the sensing system determines that an operation finishes sooner, and it synthesizes a new implementation for the operations that have not yet started or completed. First, we use an offline synthesis to determine an initial implementation considering the *wcet*s of operations. Fig. 3a shows the implementation determined offline using *wcet*. At runtime, we start to execute the application according to the offline implementation. A sensing system notifies the computer whenever an operation finishes earlier than its *wcet*. Then, ONS is run to determine a new corresponding implementation.

In this paper we propose for ONS an implementation based on List Scheduling (LS). LS-based syntheses have been proposed previously for online error recovery [1], [8], [5] and the comparisons to the approaches based on metaheuristics show that LS provides good results in a short time [1].

Our proposed ONS is presented in Fig. 5. ONS takes as input the application graph $\mathcal{G}$, the biochip architecture $\mathcal{A}$, the current implementation Ψ and the current time t. The output of ONS is an implementation $\Psi' = \{\mathcal{O}, \mathcal{B}', \mathcal{P}', \mathcal{S}', \mathcal{U}'\}$, where the allocation $\mathcal{O}$ is not changed, while new binding $\mathcal{B}'$, placement $\mathcal{P}'$, schedule $\mathcal{S}'$ and routing $\mathcal{U}'$ are decided. To obtain a better implementation with ONS, we sort offline the library in ascending order of operation execution time, i.e., the fastest modules come first in the library.

First, ONS adapts the application graph to the current execution scenario. A new graph $\mathcal{G}'$ is obtained by removing

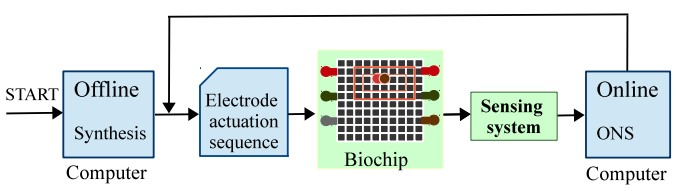

Fig. 4. The biochip setup for ONS

ONS($\mathcal{G}, \mathcal{A}, \mathcal{L}, \Psi, t$)

1: $\mathcal{G}'$ = RemoveExecutedOperations($\mathcal{G}, \Psi$)
2: CriticalPath($\mathcal{G}$)
3: *List* = GetReadyOperations($\mathcal{G}$)
4: **repeat**
5: O_i = RemoveOperation(*List*)
6: $\mathcal{P}'$ = FTP($\mathcal{L}, \mathcal{A}, O_i, t$)
7: $\mathcal{B}'$ = Bind(M_j, O_i)
8: $\mathcal{U}'$ = DetermineRoute(O_i, M_j, $\mathcal{A}$)
9: $\mathcal{S}'$ = Schedule(O_i, $\mathcal{U}'$, t, $\mathcal{L}$)
10: t = the earliest time when an operation finishes
11: UpdateReadyList($\mathcal{G}'$, t, *List*)
12: **until** *List* = $\emptyset$
13: **return** $\Psi' = \{\mathcal{O}, \mathcal{B}', \mathcal{P}', \mathcal{S}', \mathcal{U}'\}$

Fig. 5. Online synthesis strategy

the executed operations (line 1). The graph $\mathcal{G}'$ contains the operations that have not yet started or completed. Every node from $\mathcal{G}'$ is assigned a specific priority according to the critical path priority function (line 2 in Fig. 5) [10].

List contains all operations that are ready to run, sorted by priority (line 3). An operation is ready to be executed when all input droplets have been produced, i.e. all predecessor operations from the application graph $\mathcal{G}'$ finished executing. The intermediate droplets that have to wait for the other operations to finish, are stored on the biochip. Note that the operations that are interrupted in their execution at the time ONS is triggered are also included in *List*.

The algorithm takes each ready operation O_i (line 5) and performs placement, binding, routing and scheduling. For the placement of operations we have adapted the Fast Template Placement (FTP) algorithm from [2], which uses: (i) free-space partitioning manager that divides the free space in maximal empty rectangles and (ii) a search engine that selects the best-fit rectangle for each module. Hence, the function FTP (line 6) returns the first available module $M_j \in \mathcal{L}$ that can be placed on the biochip $\mathcal{A}$. Since the library is ordered by operation execution time, we know M_j is the available module that can execute O_i the fastest. Next, O_i is bound to M_j (line 7), the routing from the current placement of the input droplets to the location of M_j is determined.

Let us consider the example in Fig. 1 with the same execution scenario as in Fig. 3b, i.e., O_{10-15} finish earlier than their *wcet*. At time $t = 4$ *s*, when the mixing operation O_{10} finishes earlier than *wcet*, the computer will execute ONS to determine a new implementation. Operation O_{13} has the highest priority among all the ready operations. Module M_1 is the fastest available module (i.e., not occupied by other operations), hence O_{13} is bound to M_1.

When scheduling the operation O_i, we consider two cases: (1) O_i has not yet started executing and (2) O_i has started executing but has not yet completed (i.e., the execution of O_i was interrupted by ONS). In case (1), the operation O_i is scheduled considering the routing time overhead and the corresponding *wcet* in the module library $\mathcal{L}$. In case (2), O_i has already executed partially, hence the remaining execution

TABLE II
COMPARISON OF OFFLINE (NO VARIABILITY) APPROACH AND ONS

App.	Arch.	$\delta_{\mathcal{G}}^{OFF}(s)$	$k=30\%$		$k=50\%$		$k=70\%$	
			$\delta_{\mathcal{G}}^{ONS}(s)$	Imp.(%)	$\delta_{\mathcal{G}}^{ONS}(s)$	Imp.(%)	$\delta_{\mathcal{G}}^{ONS}(s)$	Imp.(%)
PCR	8×8 (1,1,1,0)	8.12	avg. 7.1 dev. 0.01	12.55	avg. 6.61 dev. 0.48	19	avg. 6.19 dev. 0.24	24
IVD	9×8 (1,1,1,1)	193.14	avg. 159.91 dev. 3.24	17.2	avg. 156.83 dev. 3.99	18.7	avg. 156.62 dev. 1.91	18.91
SB_1	10×11 (2,2,2,0)	36.42	avg. 31.7 dev. 0.61	12.95	avg. 30.7 dev. 1.14	19.66	avg. 28.18 dev. 1.5	22.63
SB_2	11×12 (2,2,2,2)	76.65	avg. 66.15 dev. 0.68	11.39	avg. 63.27 dev. 1.77	15.24	avg. 62.85 dev. 1.69	15.8

TABLE III
MODULE LIBRARY USED FOR EXPERIMENTS

Operation	Module area	*bcet* (s)	*wcet* (s)
Mix	2×5	1	2
Mix	2×4	2	3
Mix	1×3	3	5
Mix	3×3	5	7
Mix	2×2	7	10
Dilution	2×5	3	4
Dilution	2×4	2	5
Dilution	1×3	3	7
Dilution	3×3	6	10
Dilution	2×2	7	12
Optical detection	1×1	25	30
Dispensing	N/A	5	7

time is adjusted accordingly.

When a scheduled operation finishes executing, *List* is updated with the operations that have become ready (line 11). The repeat loop terminates when the *List* is empty (line 12).

VI. EXPERIMENTAL RESULTS

For experiments we used two synthetic benchmarks (SB_{1-2} [9]) and two real-life applications: (1) the mixing stage of polymerase chain reaction [3] (PCR, 7 operations) and (2) in-vitro diagnostics on human physiological fluids [3] (IVD, 28 operations). The deadlines for SB_{1-2}, PCR and IVD are d_{SB_1} = 85 *s*, d_{SB_2} = 80 *s*, d_{PCR} = 10 *s* and d_{IVD} = 200 *s*, respectively. The algorithms were implemented in Java (JDK 1.6) and run on a MacBook Pro computer with Intel Core 2 Duo CPU at 2.53 GHz and 4 GB of RAM. We used the experimentally determined module library from Table III.

In our experiments we were interested to determine if ONS can successfully handle variability in operation execution time. We have simulated for PCR, IVD and SB_{1-2} applications a series of scenarios where $k = 30\%$, 50% and 70% of the operations finish executing before their *wcet*. We have generated between 35 and 1000 execution scenarios considering the size of the applications and the number of operations that finish earlier than *wcet*. Table II presents the results. The biochip sizes used for each application are presented in column two. Next to the sizes, we also present in parentheses the numbers of reservoirs for the sample, buffer, reagents and optical detectors.

We are interested to determine the advantages of using ONS over the offline approach (OFF), which uses *wcet*s for the operation execution times. Thus, we have simulated the execution of PCR, IVD and SB_{1-2} on the specified architectures, and we have randomly generated an execution time between *bcet* and *wcet* for $k = 30\%$, 50% and 70% of their operations. In Table II we show the obtained average (avg.) application completion time and the mean deviation (dev.) in columns 4, 6 and 8. The reported $\delta_{\mathcal{G}}^{ONS}$ times take into account the runtime overhead required by re-synthesis (for all cases). The runtime overhead obtained on the system used for simulations varies between 10 *ms* and 270 *ms*. The mean deviation (dev.) is calculated as the average over the absolute values of deviations from the average completion time (avg.).

In columns 5, 7 and 9 we report the percentage improvement (Imp.) of ONS over OFF. The results show that ONS is a feasible alternative for the problem of variability in operation execution and obtains good results in terms of application completion time. For example, for PCR we have obtained a percentage improvement of 12.55%, 19% and 24% for $k = 30\%$, 50% and 70%, respectively.

VII. CONCLUSIONS

In this paper we have addressed the problem of variability in the execution times of the operations. We have proposed an online synthesis strategy that re-synthesizes the application at runtime whenever an operation experiences variabilities in execution time. The experiments show that ONS can take full advantage of the current configuration to derive the appropriate implementation such that the application completion time is minimized.

REFERENCES

[1] M. Alistar, P. Pop, and J. Madsen, "Online synthesis for error recovery in digital microfluidic biochips with operation variability," in *IEEE Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS*, pp. 53–58, 2012.

[2] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design and Test of Computers*, 17(1):68–83, 2000.

[3] K. Chakrabarty and F. Su, *Digital microfluidic biochips: synthesis, testing, and reconfiguration techniques*. CRC, 2006.

[4] J. Gong and C. J. Kim. All-electronic droplet generation on-chip with real-time feedback control for EWOD digital microfluidics. *Lab on a Chip*, 8(6):898–906, 2008.

[5] D. Grissom and P. Brisk. Path scheduling on digital microfluidic biochips. In *Proceedings of the 49th Annual Design Automation Conference*, pp. 26–35, 2012.

[6] O. Levenspiel. *Chemical reaction engineering*. Wiley New York, 1972.

[7] Y. Luo, K. Chakrabarty, and T.-Y. Ho, "Design of Cyberphysical Digital Microfluidic Biochips under Completion-Time Uncertainties in Fluidic Operations," in *ACM Proceedings of the 50th Annual Design Automation Conference*, pp. 44–51, 2013.

[8] Y. Luo, K. Chakrabarty, and T.-Y. Ho, "A cyberphysical synthesis approach for error recovery in digital microfluidic biochips," in *Proceedings of Design, Automation & Test in Europe Conf. & Exhibition*, pp. 1239–1244, 2012.

[9] E. Maftei, P. Pop, and J. Madsen, "Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices," *Design Automation for Embedded Systems*, 14(3):287–307, 2010.

[10] O. Sinnen, *Task scheduling for parallel systems*. Wiley-Interscience, 2007.