DTIP 16-18 April 2013, Barcelona, Spain

Operation Placement for Application-Specific Digital Microfluidic Biochips

Mirela Alistar, Paul Pop, Jan Madsen
Technical University of Denmark
Dep. of Applied Mathematics and Computer Science
DK-2800 Kgs. Lyngby, Denmark
email: mali@dtu.dk

Abstract— Microfluidic-based biochips are replacing the conventional biochemical analyzers, and are able to integrate on-chip all the necessary functions for biochemical analysis using microfluidics. The digital microfluidic biochips are based on the manipulation of liquids not as a continuous flow, but as discrete droplets on an array of electrodes. Microfluidic operations, such as transport, mixing, split, are performed on this array by routing the corresponding droplets on a series of electrodes. Researchers have proposed several approaches for the synthesis of digital microfluidic biochips. All previous work assumes that the biochip architecture is given, and consider a rectangular shape for the electrode array. However, non-regular application-specific architectures are common in practice. In this paper, we are interested in determining a placement of operations for application-specific biochips, such that the application completion time is minimized. The proposed algorithm has been evaluated using several benchmarks.

***Index Terms*—Digital Microfluidic Biochips; Computer Aided Design; Routing; Placement**

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 drug discovery, clinical diagnosis, DNA sequencing, protein analysis and immunoassays [1]–[3]. The digital microfluidic biochips (DMBs) use discrete amounts of fluids of nanoliter volume, named droplets, to perform operations such as: dispensing, transport, mixing, split and detection.

A DMB is modeled as an array of identical electrodes, where each electrode can hold a droplet. Operations such as mix and split are reconfigurable, i.e., they may take place on any of the electrodes in the array. The reconfigurable operations "execute" on "virtual devices" (also called modules) and droplets are transported on "virtual" routes, since such execution and transport can take place on any of the electrodes.

In order to be executed on a DMB, a biochemical application has to be synthesized. There is a significant amount of work on the synthesis of DMBs [3]–[5], which consists of five tasks: *allocation*, during which the needed modules are selected from a module library, *binding* the selected modules to the biochemical operations in the application, *placement*, during which the positions of the modules on the biochip are decided, *scheduling*, when the order of operations is determined and *routing* the droplets to the needed locations on the biochip.

Most researchers consider general-purpose architectures of rectangular shape. However, in practice, application-specific architectures, which are non-regular (Fig. 1) are common. For example, application-specific architectures have been proposed for disease screening in newborns [6] and for diagnostics on human physiological fluids [2]. Therefore, in [7] we have proposed an approach to the architecture synthesis of application-specific biochips. Starting from a biochemical application, modeled as a sequencing graph, and a library of physical components, our synthesis from [7] decides a physical biochip architecture which can execute the application within its specified deadline, such that the architecture cost is minimized.

In this paper we propose an approach for the placement of operations for application-specific biochips. So far, researchers have grouped the electrodes on which the fluidic operation takes place, into rectangular "virtual devices", called modules. Once a module is placed on the biochip, the corresponding electrodes are considered occupied, and cannot be used until the operation finished executing, which, in the case of application-specific biochips, results in a longer application completion time, due to inefficient use of the space. However, the operations (e.g., mixing, split) can execute anywhere on the microfluidic array, by transporting the droplets on any sequence of electrodes, forming thus a droplet route. Such a "routing-based operation execution" is proposed in [5], where operations are executed by moving the droplets freely on the biochip, without being constrained to any specific area. Such unconstrained routes are not feasible in case of contamination, a frequent problem for the biochemical assays that use proteins [8].

In this paper, we consider that the modules on which operations execute consist of circular routes, which do not have to be rectangular (see Fig. 2a for three route examples).

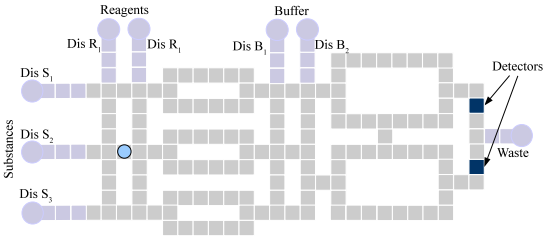


Fig. 1: Application-specific biochip architecture

 ISBN:

The circular routes are determined offline, during a pre-synthesis step, and take into account the characterization of the application-specific biochip given as input. During the operation execution, we assume that we know the positions of the droplets on the circular routes. Thus, we do not have to assume that all the electrodes of these circular-route modules are occupied. However, in case of contamination constraints, we can assign particular circular routes for specific operations. Our approach takes care that the droplets do not accidentally merge during the execution of operations.

In related literature, researchers proposed several placement strategies. In [9], a Simulated Annealing-based method is used to determine the positions of the operations on the biochip. A unified high-level synthesis and module placement, based on parallel recombinative simulated annealing, was proposed in [10]. Better results were obtained by using a T-Tree algorithm for placement [11] or by using a fast-template placement [12] integrated in a Tabu Search-based synthesis [13]. A placement approach that minimizes droplet routing, when deciding the locations of the modules, is considered in [14]. Placement strategies based on virtual topology [15] were proposed for fast synthesis approaches, which are run online for purposes such as error recovery. However, all mentioned approaches consider placement of rectangular modules, which are difficult to place on the irregular area of an application-specific biochip, resulting in unused space. Although a placement strategy for modules of non-rectangular shapes is proposed in [16], it still considers that the whole module area is occupied during the execution of the operations, blocking the corresponding electrodes from being used for other operations. An improvement is the routing-based approach from [5], which eliminates the borders of a module and allows the droplets to move freely on the biochip until the operation is completed. However, in case of contamination, the routing-based strategy requires a lot of washing, which slows considerably the execution of the bioassay and can lead to routing deadlocks.

In this paper, we propose a placement algorithm for circular-route modules, that effectively use the available area on a application specific biochip, such that the application completion time is minimized. To facilitate comparison with previous placement approaches, we integrate our placement algorithm into a List-Scheduling (LS)-based synthesis [17], which derives a complete implementation. The experiments show that our placement strategy, which considers the specificity of the architecture layout, leads to shorter application completion times.

II. System Model

A. Biochip Architecture

In a DMB, a droplet is sandwiched between a top ground electrode and a bottom electrode. The droplet is separated from electrodes by insulating layers and it can be surrounded by a filler fluid (such as silicone oil) or by air. Two glass plates, a top and a bottom one, protect the DMB from external factors. The droplets are manipulated using the electrowetting-on-dielectric (EWOD) principle [18].

On a general-purpose architecture, the electrodes are arranged to form a rectangular shape, hence the area is highly reconfigurable, as opposed to the area of an application-specific architecture, which is designed for a particular application with the intention of reducing costs. Fig. 1 shows an application-specific architecture of 164 electrodes with 7 dispensers, 1 waste reservoir and 2 detectors.

There are two types of operations: (i) non-reconfigurable, such as dispensing, executed on input reservoirs and detection, using on-chip optical detectors, and (ii) reconfigurable, such as droplet routing, mixing, dilution and split operations. A mixing operation is executed when two droplets are moved to the same location and then transported together. A split operation is done by applying concurrently the same voltage on both left and right electrodes, while the middle one remains turned off. The reconfigurable operations are executed using the physical electrodes in the biochip architecture array.

B. Biochemical Application Model

A biochemical application is modeled [1] using an acyclic directed graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where the nodes $\mathcal{V}$ represent the operations, and the edges $\mathcal{E}$ represent the dependencies between them. Let us consider part of a biochemical application, depicted as a graph in Fig. 4a, which has seven mixing operations. The directed edge between O_1 and O_3 signifies that operation O_1 has to finish before operation O_3 can start executing. Mixing operation O_3 uses the output droplet issued by operation O_1.

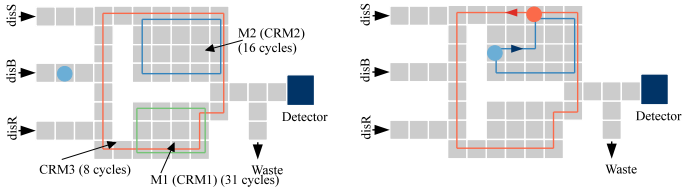
C. Circular-Route Module

Researchers have typically assumed that the electrodes used for operation execution are grouped in a rectangular area, called a "module". Based on experiments, designers characterize a library $\mathcal{L}_{\mathcal{R}}$, which contains information about the module size and corresponding execution time needed for each operation.

In this paper, we define a module as a *circular route*, of any shape including rectangular, on which droplets move repeatedly until the operation is completed (see Fig. 3c). The electrodes assigned for one module are not considered occupied, and can be used for other operations. As a consequence, the routes for different operations may overlap over several electrodes. In order to avoid undesired droplet merging for intersecting routes during runtime, we instruct one of the droplets to take a detour from its predetermined route as shown in Fig. 2b or to wait until the other droplet passed by. Some biochemical applications use protein-based compounds that can leave residues behind. To avoid such contamination, we minimize the number of overlapping electrodes among the circular-route modules used by operations that have specified contamination conflicts. By doing so, we restrict the washing operations to the overlapping electrodes, thus minimizing the execution overhead due to contamination avoidance.

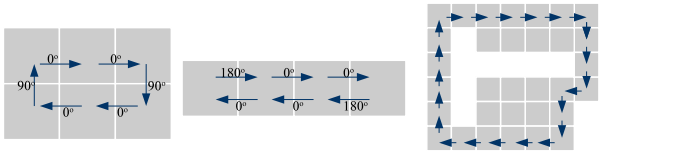
We use the routing-based operation execution proposed in [5] to estimate the operation completion time for each such circular route. In [5], the droplet movement during an operation is decomposed into basic moves and the impact of each basic

 ISBN:



a. Application-specific architecture b. Adjusted route to avoid droplet merging

Fig. 2: Example of circular-route modules



a) 2x3 module, t=6.1 s b) 1x4 module, t=4.6 s c) module of circular route, t=2.27 s

Fig. 3: Routing-based operation execution

move on the operation execution is calculated. As seen in Fig. 3b, on a 1×4 mixer, the droplets complete one cycle in 3 moves: one of 180° followed by two moves of 0°. On a 2×3 mixer (see Fig. 3a), a cycle is completed by forward moves (0°), followed by turns (90°). Using an experimentally determined library, that contains information about the execution times of the operations, the method proposed in [5] estimates, for each move, the percentage completion towards operation execution. Thus we can determine n—the minimum number of times the droplets have to rotate on a given route to achieve at least 100% operation completion. The total execution time is obtained by multiplying n with the time needed to complete one rotation. For example, for the route depicted in Fig. 3c, the droplets need to cycle 10 times in order to complete the mixing operation, resulting in an execution time of $t = 2.27$ s.

III. Problem Formulation

Let us illustrate the complete synthesis process by considering the application graph $\mathcal{G}$ depicted in Fig. 4a, consisting of seven mixing operations, that have to execute on the biochip architecture $\mathcal{A}$ from Fig. 2a. In order to synthesize the application $\mathcal{G}$ on the given biochip $\mathcal{A}$, we need to first bind each operation to a module and then, during placement, to find a location for the selected modules on the biochip.

Let us first consider that we can use only rectangular modules. For the considered application $\mathcal{G}$, we select from the rectangular module library[1] $\mathcal{L}_R$, the 3×4 rectangular module M_1 for operations O_1, O_5 and O_6 and of the 4×5 rectangular module M_2 for O_2, O_4, O_3 and O_7. Next, we have to find available space on the biochip to place M_1 and M_2 so that they do not overlap. The optimal placement solution is shown in Fig. 2a. After the placement is done, we schedule the operations, i.e. we determine the order in which they are executed. The schedule is illustrated in Fig. 4b, as a Gantt chart, where the operations are presented as rectangles with the length equal to their duration, measured in seconds. As shown in Fig. 4b, for the case when rectangular modules are used, we obtain a completion time of $t = 11.43$ s.

However, an application-specific architecture has an irregular shape, which makes placement of rectangular modules difficult. As a direct consequence, a lot of biochip area remains unused. For the same application and biochip architecture, if we consider circular-route modules an application execution time of $t = 8.72$ s is obtained. As depicted by the schedule from Fig. 4c, operations O_4, O_5, O_6 and O_7 execute on the additional circular-route module CRM_3 (see Fig. 2a for the CRMs and Table I for the corresponding execution times), in parallel with the other operations, thus decreasing the total execution time of the application. We ignored routing in this example, for simplicity reasons. For this example, using a circular-routes, instead of only rectangular modules, resulted in an improvement of 23% in the application completion time. The improvement is due to circular-routes, which make better use of the area and the characteristics of the considered application-specific architecture.

We formulate our operation placement problem as follows: given as input a biochemical application modeled as a graph $\mathcal{G}$ and an application-specific architecture $\mathcal{A}$, we are interested to determine the circular route module for each operation, and its placement on $\mathcal{A}$, such that the application completion time $\delta_{\mathcal{G}}$ is minimized.

[1]We use the library from [19] for this example.

TABLE I: Example of library for circular-route modules

Operation	Circular-route module	Operation time (s)
Mix	CRM_1	3.05
Mix	CRM_2	2.28
Mix	CRM_3	2.18

IV. Placement of Circular-Route Modules

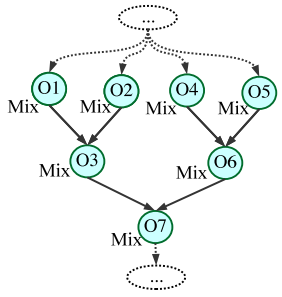
We propose a two-step strategy for our problem. During the first step, we determine a library $\mathcal{L}$ of circular-route modules (see Sections IV-B and IV-C). In the second step, we select from $\mathcal{L}$, during synthesis, a circular-route module for the current operation to be executed, such that the completion time $\delta_{\mathcal{G}}$ is minimized. Note that the library $\mathcal{L}$, determined at step one, can be used with any synthesis approach.

A. List-Scheduling Synthesis

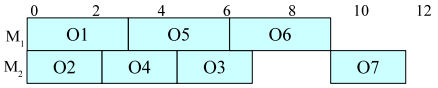
The synthesis process is a NP-complete problem for which several approaches have been proposed. Metaheuristics, such as [3], [16] are able to obtain near-optimal results for application completion time $\delta_{\mathcal{G}}$, but take a long time. Recently, for the synthesis of DMBs, List Scheduling (LS)-based heuristics [17] have been proposed, which are faster, but cannot guarantee the optimality. For step two, we use the LS-based synthesis from [7].

As illustrated in Fig. 5, the LS-based synthesis takes as input the application-specific architecture $\mathcal{A}$, the application graph $\mathcal{G}$, and the library of circular-route modules $\mathcal{L}$ and outputs the application completion time $\delta_{\mathcal{G}}$.

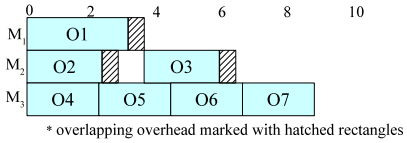
Every node from $\mathcal{G}$ is assigned a specific priority according to the critical path priority function (line 1) [20]. *List* contains all operations that are ready to run, sorted by priority.



a. Biochemical application subgraph



b. Schedule obtained using rectangular modules



* overlapping overhead marked with hatched rectangles

c. Schedule obtained using circular-route modules

Fig. 4: Biochemical application and synthesis results

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. For each ready operation O_i, stored in *List*, **Place** (line 5) searches in the library $\mathcal{L}$ for the fastest circular-route module that can be placed on the biochip at current time t. If such a circular-route module is found, the operation O_i is scheduled at time t. When an operation finishes executing, the *List* is updated with the operations that have become ready (line 8). The **ListScheduling** outputs the schedule table obtained for $\mathcal{G}$. The application completion time $\delta_{\mathcal{G}}$ is the finishing time of the last operation in the schedule table.

Let us assume that we have to synthesize the graph from Fig. 4a on the application-specific biochip from Fig. 2a, using the circular-route module library $\mathcal{L}$ from Table I. Next, let us assume that, at time $t = 4.36$ s dilution operation O_6 has the highest priority among all the ready operations (an operation is ready if all its input droplets have arrived). For O_6, **Place** (line 5) selects from library $\mathcal{L}$ (Table I), the circular-route module CRM_3 (see Fig. 2a), since it can be placed on the biochip and it finishes the mixing operation the fastest. At time $t = 6.54$ s, the dilution operation O_6 finishes executing (line 7), and *List* is updated (line 8) with its successor, the mixing operation O_7, which becomes ready to execute. For this example, the LS-based synthesis will produce the schedule table from Fig. 4c.

B. Building a Library of Circular-Route Modules

During the first step of our strategy, we build a library $\mathcal{L}$ of circular-route modules for the application-specific architecture $\mathcal{A}$ (see Fig. 6). We want to determine circular-route modules that will use effectively the area on $\mathcal{A}$, so that the application completion time is minimized. Mixing of two droplets is achieved faster when the forward movement of the droplets

ListScheduling($\mathcal{G}$, $\mathcal{A}$, $\mathcal{L}$)

```
1  CriticalPath(G)
2  repeat
3     List = GetReadyOperations(G)
4     O_i = RemoveOperation(List)
5     if Place(O_i, A, L) then
6        t_i^start = Schedule(O_i, A)
7        t = earliest time when an operation terminates
8        UpdateReadyList(G, t, List)
9     end if
10 until List = ∅
11 return δ_G
```

Fig. 5: List Scheduling synthesis

is increased and the backward movement is avoided [19]. An application-specific architecture can have an irregular shape (see Fig. 1), so we need to find locations where operations can be favorably executed.

BuildLibrary (Fig. 6) starts by identifying restricted rectangles (RRs) (line 2), which are areas of rectangular shape bordered by the margins of the architecture. We use the cutting algorithm from [21], developed for paper cutting problems, where the material needs to be optimally cut so that it minimizes waste. The list of restricted rectangles L_{RR} (line 2) is obtained by using "guillotine" cuts, done parallel with the edges of the architecture. We start the cuts from each corner-electrodes of the architecture, using horizontal and vertical cuts. A corner-electrode is an electrode that at least two edges which are not bordered by any other electrode. To obtain R_{XY} we cut horizontally and vertically, and then, changing the order, we use first vertical and then horizontal cuts to obtain R_{YX}, like the RRs depicted in Fig. 8 obtained for the bottom right corner of the architecture. In some of the cases, R_{XY} is the same rectangle as R_{YX}. Also, we consider those unused areas containing inactive electrodes and we extend L_{RR} with restricted rectangles of such inactive electrodes. In this case, the restricted rectangles will be bordered by active electrodes, see $R_{XY}(inactive)$ in Fig. 8. We use the RRs as guiding areas for obtaining CRMs. Thus, for each RR found, we determine a list of circular route modules $\mathcal{L}_{CRM}$ (line 4), which is stored in the library $\mathcal{L}$.

As an input to the **DetermineCRM** function (presented in Fig. 7 and discussed in Section IV-C), we use the control parameters *MinR, MaxR, MinW, MaxW*, which are experimentally determined for a given application-specific architecture. The circular-route modules are stored in the library $\mathcal{L}$ and used during the synthesis. It is difficult to predict at the pre-synthesis stage which circular-route should be selected, to reduce the $\delta_{\mathcal{G}}$ of the application, as it depends not only on the architecture,

BuildLibrary($\mathcal{A}$, *MinR*, *MaxR*, *MinW*, *MaxW*)

```
1  L - the library of Circular-Route Modules (CRM)
2  L_RR = DetermineRestrictedRectangles(A)
3  for each RR_i in L_RR do
4     L_CRM = DetermineCRM(A, RR_i, MinR, MaxR, MinW, MaxW)
5     InsertInLibrary(L_CRM, L)
6  end for
7  return L
```

Fig. 6: Building the library $\mathcal{L}$

ISBN:

DetermineCRM($\mathcal{A}, RR, MinR, MaxR, MinW, MaxW$)

1 L_{CRM} = List of circular route modules
2 FillArch($\mathcal{A}$, RR)
3 L_{SP} = GetStartPosition($\mathcal{A}$, RR, $Radius$)
4 **for each** $StartPos$ in L_{SP} **do**
5 **for** $Radius$ from $MinR$ to $MaxR$ **do**
6 **for** $Window$ from $MinW$ to $MaxW$ **do**
7 CRM = new circular route module
8 **repeat**
9 $NextPos$ = GetBestNeighbor($CRM, Radius, Window$)
10 InsertInRoute($CRM, NextPos$)
11 **until** $NextPos$ is $StartPos$
12 UpdateList(L_{CRM}, CRM)
13 **end for**
14 **end for**
15 **end for**
16 InsertInList(L_{CRM}, RR)
17 **return** L_{CRM}

Fig. 7: Algorithm to determine circular-route modules

but also on application's particularities such as dependencies between operations and contamination constraints. Storing all possible circular-route modules in the library $\mathcal{L}$ is an expensive solution for cases when a fast synthesis is required. Hence, we determine for each restricted rectangle a fixed number of circular-route modules L_{CRM} (line 7) for an operation to be selected during synthesis.

C. Determining a Circular-Route Module

For each restricted rectangle (RR), we determine a list of circular-route modules L_{CRM} using **DetermineCRM**, illustrated in Fig. 7. We start from the centroid (geometric center) of the RR and "graphic fill" the architecture, considering each electrode a pixel (line 2). The centroid of a rectangle is situated at the intersection of its diagonals. Fig. 8 shows a filled architecture starting from the centroid of the restricted rectangle R_1. We use a greedy approach to find circular-route modules that fulfill the distance constraints set by control parameters *MinR, MaxR, MinW, MaxW* (line 8). *MinR* and *MaxR* bound the Radius, which sets the distance from the centroid and it is used to determine the start position of the circular-route module. *MinW* and *MaxW* set the boundaries for the next electrodes of the circular-route module, which can be situated at a distance from the center that variates between [*Radius* - *Window*, *Radius*], where *Window* can have any value in the range [*MinW*, *MaxW*].

After the architecture is filled, the list of start points L_{SP} for the circular-route module is determined (line 3), by selecting all points located at a distance equal with *Radius* from the centroid of the considered RR. For example, for the restricted rectangle R_1 from Fig. 8, and a $Radius = 6$, the list of starting points L_{SP} contains all electrodes marked with 6. From each of the starting points we construct a route (line 6–11), by adding new electrodes until the route completes in a circle, i.e., it reaches the starting point.

GetBestNeighbor (line 8) uses a greedy randomized [22] approach to select the next electrode of the route. Out of the possible next electrodes, which are those that can be reached from the current position, **GetBestNeighbor** selects from the neighbors that are located within the boundaries imposed by control parameters, the one that leads to the largest operation completion percentage (see Section II-C). In case there are two equally good candidates for the next position, **GetBestNeighbor** randomly selects one of them. Backward moves are also permitted in case there are no other options. Let us consider the current position *Pos*, marked in Fig. 8, and the given control parameters $Radius = 6$, $MinW = 1$ and $MaxW = 3$. For such a window of size 3, out of the four neighboring cells that can be reached from *Pos*, only three (the left, up and down ones) fulfill the distance requirement, which is to be at a distance between 6 and 3 from the centroid of R_1. In our example, selecting the top or bottom electrode improves the mixing with 0.1%, while the left electrode with -0.5%, due to flow reversibility. We randomly select the top electrode to be the next position. For each restricted rectangle RR we determine a number N_{CRM} of circular-route modules, which depends on the control parameters: $N_{CRM}(RR) = \sum_{Radius=MinR}^{MaxR} ((MaxW - MinW + 1) \times E_{Radius})$, where E_{Radius} is the number of electrodes situated at *Radius* distance from the centroid of RR. Each of the N_{CRM} circular-route modules is evaluated, and only three are stored in L_{CRM} (line 12): the one that minimizes the use of area, the one that minimizes operation completion time; a third circular-route module, represented by the corresponding RR is also stored in L_{CRM} (line 16). **DetermineCRM** returns the list of circular-route modules L_{CRM} (line 17).

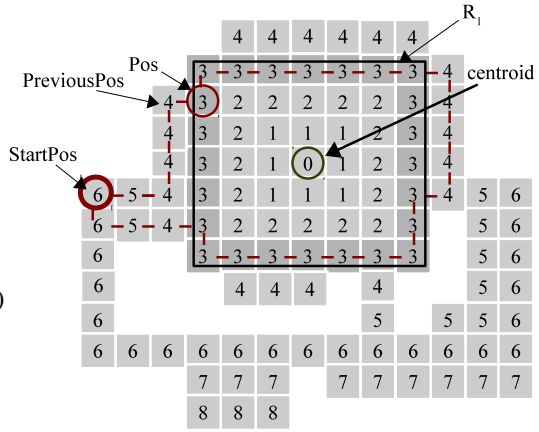


Fig. 8: Determining circular-route modules

V. Experimental Results

For experiments we used one real-life application [3]: in-vitro diagnostics on human physiological fluids (IVD, 28 ops.) and three synthetic benchmarks (SB_1 to SB_3). The algorithms were implemented in Java (JDK 1.6) and run on a MacBook Pro with Intel Core 2 Duo CPU at 2.53 GHz and 4 GB of RAM.

In our experiments we were interested to determine the efficiency of our proposed placement of operations (Section IV)

TABLE II: Experimental results

App. (ops.*)	Arch.	*MinR, MaxR*	$\delta^R_{\mathcal{G}}(s)$	$\delta^{CRM}_{\mathcal{G}}(s)$	Deviation (%)
IVD (23)	45 (2,2,2)	[3,5]	11.73	18.4	36
SB_1 (50)	96 (1,2,1)	[3,5]	29.39	23.9	18.6
SB_2 (70)	103 (2,2,2)	[3,6]	31.03	20.15	35
SB_3 (90)	125 (2,2,2)	[3,8]	42.51	27.87	34

*We ignored the detection operations for experiments.

in terms of the application completion time $\delta^{CRM}_{\mathcal{G}}$ obtained after synthesis. We compared $\delta^{CRM}_{\mathcal{G}}$ to the completion time $\delta^R_{\mathcal{G}}$, obtained by using the routing-based synthesis approach from [5], which is the only available synthesis approach that is not limited to rectangular modules and can take advantage of an application-specific architecture. The results of this comparison are presented in Table II. The size of the used benchmarks is presented in column 1 of each table, and the control parameters *MinR, MaxR*, used for building the circular route library (Section IV-B) are presented in column 3 (we used $MinW = 1$ and $MaxW = 3$ for all the experiments). For the real-life application (IVD), we used the application-specific architecture (column 2) derived with our architecture synthesis from [7]. The application-specific architectures for the synthetic benchmarks were obtained manually. In column 2 we present, for each architecture, the number of electrodes and in parentheses the numbers of reservoirs for sample, buffer and reagent. As we can see from Table II, our placement results in a better completion time $\delta^{CRM}_{\mathcal{G}}$ (column 5) than $\delta^R_{\mathcal{G}}$ (column 4) for all the tested benchmarks. For example, for IVD, we obtained a completion time $\delta^{CRM}_{\mathcal{G}} = 11.73$ s, improving with 36% the completion time $\delta^R_{\mathcal{G}} = 18.4$ s.

In all our experiments the capacity of a circular-route modules was set to 1, i.e., maximum one operation could use the circular-route module at a specific time. This was done to limit the potential contamination. However, circular-route modules were allowed to intersect under the assumption that wash droplets are used at intersection points where contamination is an issue.

VI. Conclusions

We have proposed a strategy for the placement of operations on application-specific DMB architectures, such that the application completion time is minimized. We build a library of circular-route modules that take advantage of the characteristics of the architecture and use effectively the available area. The library provides multiple choices, that can be further exploited by the synthesis implementation to fulfill requirements such as contamination constraints or fast completion time. We have evaluated our proposed placement strategy, by comparing the synthesis results with previous work, on several benchmarks.

As the experimental results show, our placement strategy is able to significantly reduce the completion time of the applications. We plan to integrate the proposed placement into our architecture synthesis heuristics from [7], which determines a specific architecture for a given application.

References

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

[2] V. Srinivasan, V. K. Pamula, and R. B. Fair, "An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids," *Lab Chip*, vol. 4, no. 4, pp. 310–315, 2004.

[3] K. Chakrabarty, R. B. Fair, and J. Zeng, "Design tools for digital microfluidic biochips: toward functional diversification and more than moore," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 7, pp. 1001–1017, 2010.

[4] T.-W. Huang, S.-Y. Yeh, and T.-Y. Ho, "A network-flow based pin-count aware routing algorithm for broadcast electrode-addressing ewod chips," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 425–431.

[5] E. Maftei, P. Pop, and J. Madsen, "Routing-based synthesis of digital microfluidic biochips," in *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2010, pp. 41–50.

[6] R. S. Sista, A. E. Eckhardt, T. Wang, C. Graham, J. L. Rouse, S. M. Norton, V. Srinivasan, M. G. Pollack, A. A. Tolun, D. Bali *et al.*, "Digital microfluidic platform for multiplexing enzyme assays: implications for lysosomal storage disease screening in newborns," *Clinical chemistry*, vol. 57, no. 10, pp. 1444–1451, 2011.

[7] M. Alistar, P. Pop, and J. Madsen, "Application-specific fault-tolerant architecture synthesis for digital microfluidic bioships," in *Asia and South Pacific Design Automation Conference, 2013*. IEEE, 2013.

[8] Y. Zhao and K. Chakrabarty, "Cross-contamination avoidance for droplet routing," in *Design and Testing of Digital Microfluidic Biochips*. Springer, 2013, pp. 27–55.

[9] F. Su and K. Chakrabarty, "Module placement for fault-tolerant microfluidics-based biochips," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, pp. 682–710, 2006.

[10] ——, "Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips," in *Design Automation Conference, 2005. Proceedings. 42nd*. IEEE, 2005, pp. 825–830.

[11] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Placement of defect-tolerant digital microfluidic biochips using the t-tree formulation," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 3, no. 3, p. 13, 2007.

[12] K. Bazargan, R. Kastner, M. Sarrafzadeh *et al.*, "Fast template placement for reconfigurable computing systems," *IEEE Design & Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[13] E. Maftei, P. Paul, and J. Madsen, "Tabu search-based synthesis of dynamically reconfigurable digital microfluidic biochips," *ACM CASES*, pp. 195–203, 2009.

[14] T. Xu and K. Chakrabarty, "Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 4, no. 3, p. 11, 2008.

[15] D. Grissom and P. Brisk, "Fast online synthesis of generally programmable digital microfluidic biochips," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2012, pp. 413–422.

[16] 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*, vol. 14, no. 3, pp. 287–307, 2010.

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

[18] M. G. Pollack, R. B. Fair, and A. D. Shenderov, "Electrowetting-based actuation of liquid droplets for microfluidic applications," *Applied Physics Letters*, vol. 77, no. 11, pp. 1725–1726, 2000.

[19] P. Paik, V. K. Pamula, and R. B. Fair, "Rapid droplet mixers for digital microfluidic systems," *Lab Chip*, vol. 3, no. 4, pp. 253–259, 2003.

[20] G. De Micheli *et al.*, *Synthesis and optimization of digital circuits*. McGraw-Hill New York, 1994, vol. 94.

[21] U. Twisselmann, "Cutting rectangles avoiding rectangular defects," *Applied mathematics letters*, vol. 12, no. 6, pp. 135–138, 1999.

[22] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.