Placement-Aware Architectural Synthesis of Digital Microfluidic Biochips using ILP

Elena Maftei, Paul Pop, Jan Madsen, Thomas Stidsen
Technical Univ. of Denmark, DK-2800 Kgs. Lyngby
{em|pop|jan|tks}@imm.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 (hence the term *digital*), and thus are highly reconfigurable and scalable. We model a biochemical application using an abstract model consisting of a sequencing graph. The digital biochip is modeled as a two-dimensional array of cells, where each cell can hold a droplet. In this paper we propose an integer linear programming (ILP) synthesis methodology that determines the allocation, resource binding, and scheduling of the operations in the application (architectural synthesis) at the same time with module placement (physical synthesis). Although architectural and physical synthesis steps are typically performed separately, we show that significant improvements can be gained by considering the placement during architectural synthesis. To handle large problem sizes, we have extended the ILP implementation with *Local Branching*, which is a meta-heuristic for design-space exploration that uses the ILP solver to perform the local searches. The proposed methodology has been evaluated using several real-life examples.**

I. Introduction

Microfluidic-based biochips (also referred to as lab-on-a-chip) are replacing the conventional biochemical analyzers, and are able to integrate on-chip all the necessary functions for biochemical analysis using microfluidics, such as, transport, splitting, merging, dispensing, mixing, and detection. Applications areas of biochips include: clinical diagnostics, bio-defense applications, massively parallel DNA analysis and automated drug discovery [1]. Biochips are able to: provide miniaturization, thus enabling very small volumes and speeding up chemical reactions and analytical detection; obtain higher throughput with minimal human intervention; use smaller sample and reagent consumption; provide higher sensitivity at significantly lower costs per assay than the traditional methods; and increase productivity through automation and parallelization [1].

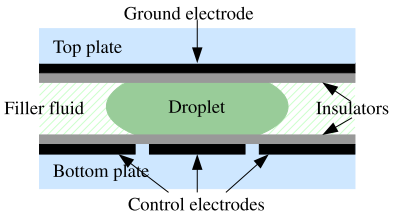
There are two approaches to microfluidics. The "first generation" is based on the continuous flow of liquid through micro-channels using micropumps and microvalves [2] The "second generation" is based on the manipulation of liquids not as a continuous flow, but as discrete droplets [3]. Although the continuous-flow biochips have been used for simple biochemical applications, due to their lack of flexibility they are unsuitable for more complex applications that require complicated fluid manipulations [4]. Therefore, in this paper, we are interested in droplet-based digital biochips, which are highly reconfigurable and scalable.

CAD tools for digital microfluidic biochips (DMBs) are in their infancy, and designers are using manual, bottom-up, full-custom, design approaches to implement such biochips [5]. However, DMBs are becoming increasingly complex, and are expected to be integrated with microelectronic components in next generation system-on-chips. Consequently, the current bottom-up full-custom design approach will not scale to the new designs. Therefore, new top-down methods and techniques are required, which can offer the same level of support as the one taken for granted currently in the semiconductors industry [5].

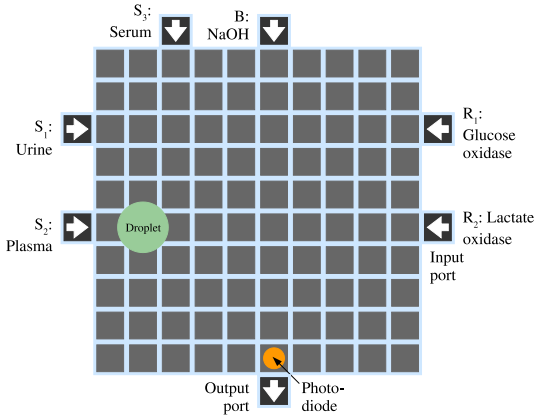
Researchers have initially addressed separately architectural-level and physical-level synthesis of DMBs. Su and Chakrabarty [6] have proposed an ILP model for scheduling and binding, considering a given allocation, and without addressing placement and routing. During the physical-level synthesis, the placement [7] of each module on the microfluidic array and the routing [8], [9] of droplets from one module to another have to be determined. A unified high-level synthesis and module placement methodology has been proposed in [10], where the focus has been on deriving an implementation that can tolerate faulty cells in the biochip array. Their algorithm has later been improved by Yuh et al [11]. However, the placement has been done without considering that droplets require additional on-chip space for routing.

The combined architectural- and physical-synthesis problem has similarities with the offline configuration management of dynamically reconfigurable FPGAs [12] and in particular with that of 3D module placement. Whereas, offline 3D placement algorithms aim at packing operations as densly as possible [13], our goal is to derive an implementation that minimizes the completion time of the biochemical application, given an area constraint.

In this paper we propose an integer linear programming (ILP) synthesis methodology that, starting from a biochemical application modeled as a sequencing graph and a given biochip array, determines the allocation, resource binding, and scheduling of the operations in the application at the same time with module placement. The placement is performed such that the droplets are routable. ILP is a generic, expressive and extensible approach that relies on sophisticated solvers to produce provable optimal solutions. However, the optimality often comes at a very expensive computational cost. To handle large problem sizes within the ILP framework, we have used *Local Branching* [14], which is a meta-heuristic for design-space exploration that uses the ILP solver to efficiently perform the local searches.



(a) Cell architecture



(b) Biochip: array of cells

Operation	Area (cells)	Time (s)
Mixing	2x2	6
Mixing	2x3	5
Mixing	2x4	4
Dilution	2x2	6
Dilution	2x3	5
Dilution	2x4	4
Storage	1x1	–
Detection	–	8

(c) Module library

Fig. 1: Biochip architecture

The paper is organized in six sections. Sections II-A and II-B present the model of the digital microfluidic biochip and the sequencing graph model we use to capture a biochemical application, respectively. We formulate the problem in Section III and illustrate the design tasks using several examples. The proposed ILP model is presented in Section IV. The evaluation of the proposed approach is performed in Section V. The last section presents our conclusions.

II. System Model

A. Digital Microfluidic Biochip Architecture

In a digital microfluidic biochip the manipulation of liquids is performed using discrete droplets. There are several mechanisms for droplet manipulation [15]. Our proposed research will consider electrowetting-on-dielectric (EWD) [3], but can be extended to handle other techniques as well. EWD is the most promising technique, and can provide high droplet speeds of up to 20 cm/s.

A biochip is composed of a two-dimensional array of cells. The schematic of a cell is presented in Figure 1(a). The droplet is sandwiched between two glass plates (the top plate and the bottom plate), and moves within a filler fluid. The top plate contains a single ground electrode, while the bottom plate has several control electrodes. The electrodes are insulated from the droplet trough an insulation material. With EWD, the movement of droplets is controlled by applying voltages to the required electrodes. For example, turning off the middle control electrode and turning on the right control electrode in Figure 1(a) will force the droplet to move to the right. For the details of the EWD-based chip fabrication, the reader is directed to [3].

Several cells are put together to form a two-dimensional array (an example architecture is presented in Figure 1(b)). Using EWD manipulation, droplets can be moved to any location without the need for pumps and valves, which are required in a continuous-flow biochip. Besides the basic cell discussed previously, the chip typically contains input and output ports and detectors. The detection can be done by using, for example, a LED beneath the bottom plate and a photodiode on the top plate.

Using this architecture, and changing correspondingly the control voltages, several operations, such as transport, splitting, merging, dispensing, mixing, and detection, can be performed. For example, mixing is done by transporting two droplets to the same location, and then moving them next to each other on a circular path within a delimited cell block, until they mix. Any cells in the chip can be used for such an operation, thus, we say that the chip is "reconfigurable".

As is the case with digital circuits, we consider that designers will build and characterize a module library $\mathcal{L}$, where for each operation there are several options varying in terms of area and execution time, see Figure 1(c).

B. Biochemical Application Model

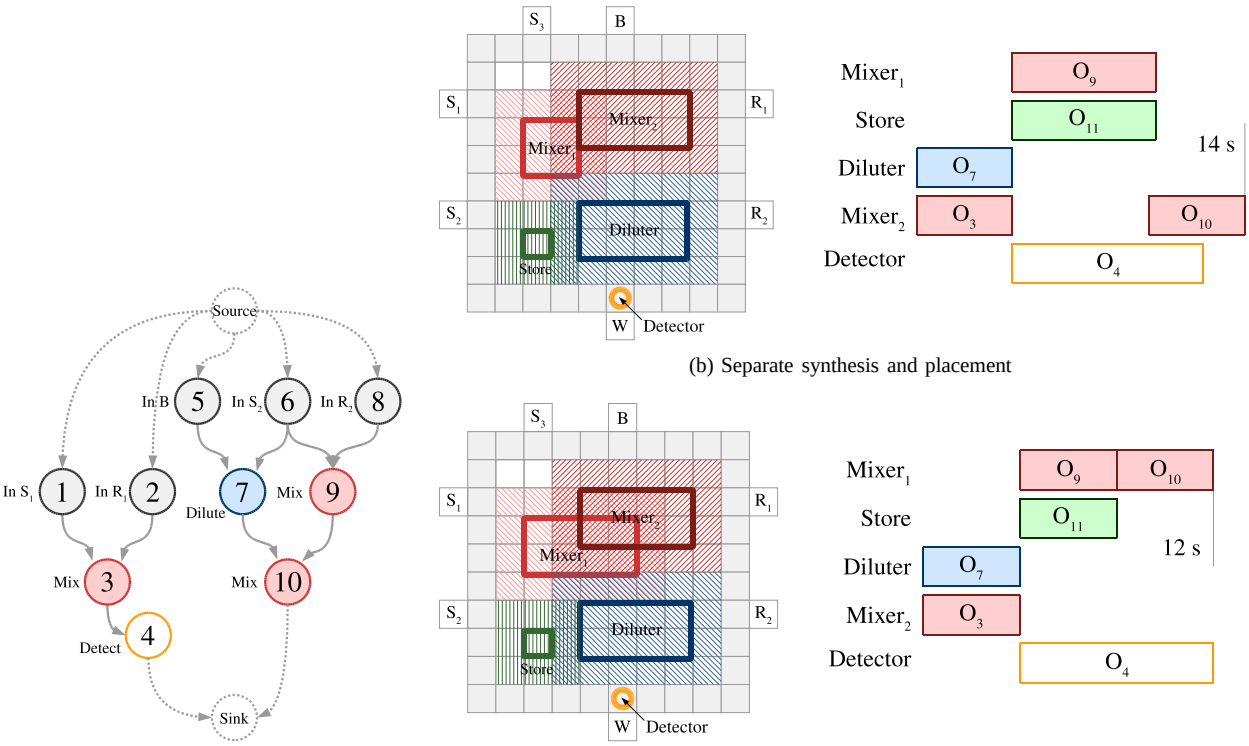
We model a biochemical application using an abstract model consisting of a sequencing graph [5]. The graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ is directed, acyclic and polar (i.e., there is a *source node*, which is a node that has no predecessors and a *sink node* that has no successors). Each node $O_i \in \mathcal{V}$ represents one operation. The binding of operations to modules in the architecture is captured by the function $\mathcal{B}: \mathcal{V} \rightarrow \mathcal{A}$, where $\mathcal{A} \subset \mathcal{L}$ is the set of allocated modules from the given library $\mathcal{L}$.

An edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of operation O_i is the input of O_j. An operation can be activated after all its inputs have arrived and it issues its outputs when it terminates. Operations are non-preemptable and thus cannot be interrupted during their execution.

We assume that, for each operation O_i, we know the execution time $C_i^{M_k}$ on module $M_k = \mathcal{B}(O_i)$ where it is assigned for execution. Currently, the routing time between two operations is an order of magnitude smaller compared to the operation time. Hence, we consider the routing time to be part of the operation execution time and do not model it explicitly.

III. Problem Formulation

The problem we are addressing in this paper can be formulated as follows. Given (1) a biochemical application modeled as a graph $\mathcal{G}$, (2) a biochip consisting of a two-dimensional



(a) Example application

(b) Separate synthesis and placement

(c) Unified synthesis and placement

Fig. 2: Implementation example

$m \times n$ array of cells, and (3) a characterized module library $\mathcal{L}$, we are interested to synthesize that implementation Ψ, which minimizes the completion time of the application (i.e., finishing time of the sink node, t_{sink}^{finish}).

Synthesizing an implementation $\Psi = < \mathcal{A}, \mathcal{P}, \mathcal{B}, \mathcal{S} >$ means deciding on: (1) the allocation $\mathcal{A} \subset \mathcal{L}$, which determines what modules from the library $\mathcal{L}$ should be used, (2) the placement $\mathcal{P}$ of the modules on the $m \times n$ array, (3) the binding $\mathcal{B}$ of each operation $O_i \in \mathcal{V}$ to a module $M_k \in \mathcal{A}$, and the schedule $\mathcal{S}$ of the operations, which contains the start time t_i^{start} of each operation O_i on its corresponding module. The next subsections will illustrate each of these subproblems.

A. Allocation and placement

Let us consider the application graph $\mathcal{G}$ in Figure 2(a), where we have ten operations, O_1 to O_{10}. We would like to implement this application on the 10x10 biochip from Figure 1(b). The input and detection operations are already assigned to the corresponding input ports and detection module, respectively. Thus, O_1 is assigned to the input port S_1, O_2 to R_1, O_5 to B, O_6 to S_2 and O_8 to R_2. The detection operation O_4 will be performed by the on-chip detector, and then the droplet will be moved to the waste reservoir through the output port W. However, for the mixing operations (O_3, O_9 and O_{10}) and the dilution operation O_7 our synthesis approach will have to allocate the appropriate modules.

Let us assume that the available module library is the one captured by the table in Figure 1(c). We have to select those modules that will lead to the minimum application completion time and place them on the 10x10 chip. Figure 2(c) presents the allocation and placement. We use the following modules: two 2x4 mixers, one 2x4 diluter and one 1x1 "store" module.

Note that special "store" modules have to be allocated if a droplet has to wait before being processed. Consider the mixing operation O_{10}, which mixes two droplets, one from O_7 and one from O_9. After O_7 finishes, a 1x1 storage cell is required to store the droplet before O_{10} starts. In general, if there exists an edge $e_{i,j}$ from O_i to O_j such that O_j is not immediately scheduled after O_i (i.e., there is a delay between the finishing time of O_i and the start time of O_j) then we will have to allocate a storage cell for $e_{i,j}$. Hence, the allocation of storage cells depends on how the schedule is constructed.

The placement for the discussed solution is as indicated in Figure 2(c), where we can notice that modules occupy a space larger than their size (the hashed area corresponding to each module). This is to avoid droplet-merging and contamination. If two droplets are next to each other on two adjacent cells, they will tend to merge to form one single droplet. Therefore, we consider for each module a border of one-cell size. For example, $Mixer_1$ which has a size of 2x4 will occupy 4x6 cells. The borders also guarantee that the droplets are routable. In addition, to allow routing from and to the input and output ports, we also reserve cells for routing along the chip borders. Hence, the usable chip area in our case is 8x8.

The main difference between our placement problem and the placement for microelectronic chips [16] is that, in our case, modules can physically overlap on-chip as long as they do not overlap in time, i.e., they are used during different time intervals. This property is due to the reconfigurability of the

digital microfluidic biochip. After an operation has finished executing on a module, we can reuse the same cells as part of another module.

B. Binding and Scheduling

Once the modules have been allocated and placed on the cell array, we have to decide where to execute the operations (binding) and in which order (scheduling), such that the application completion time is minimized.

Considering the graph in Figure 2(a), Figure 2(c) presents the optimal schedule. 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. For example, operation O_9 is bound to module $Mixer_1$ (i.e., $\mathcal{B}(O_9) = Mixer_1$). O_9 starts immediately after the dilution operation O_7 (i.e, $t_9^{start} = 4$) and takes 4 s, finishing at time $t_9^{finish} = 8s$. The total schedule length will be 12 s. We consider that the schedule is divided in time-steps of one second, and we capture the set of time-steps with $\mathcal{T}$. Note that a new operation has been introduced, O_{11}, which corresponds to the storing of the second droplet before undergoing detection.

The examples so far have illustrated the optimal solution for the unified architectural synthesis and placement problem. If the architectural synthesis phase is considered separately from the placement, we get the allocation as in Figure 2(b), where three modules are used: a 2x2 mixer (4x4 with border), a 2x4 mixer (4x6 with border) and a 2x4 diluter (4x6 with border). Since 4x4 + 4x6 + 4x6 = 64, the architectural synthesis will wrongly assume that the modules can be placed concurrently on the array (which has a usable area of 8x8 = 64). This would parallelize operations O_3 (on $Mixer_2$), O_7 (on $Diluter$) and O_9 (on $Mixer_1$) and lead to the best schedule. However, the modules cannot be placed on the chip without overlapping, unless they are separated in time. This will lead to the schedule in Figure 2(b), where O_9 on $Mixer_1$ has to be delayed to not to overlap with $Mixer_2$ and $Diluter$.

IV. ILP-Based Synthesis

The problem presented in the previous section is NP-complete (scheduling in even simpler contexts is NP-complete [17]). We have developed an ILP model, and we use an ILP solver to obtain those implementations that minimize the schedule length under the imposed constraints.

In an ILP model a system is described by a minimization objective and a set of constraints which define valid conditions for the system variables. A solution to the modeled problem is an enumeration of all system variables, such that the constraints are satisfied. The optimization objective is specified as minimizing the completion time of the application,

$$\text{minimize } t_{sink}^{finish}, \tag{1}$$

where t_{sink}^{finish} is the finishing time of the sink node of the application.

The constraints fall under the following categories: (i) scheduling and precedence, (ii) resource and (iii) placement constraints. In order to be able to express them, a binary variable is defined as follows:

$$z_{i,j,k,l} = \begin{cases} 1, & \text{if operation } O_i \text{ starts executing at} \\ & \text{time-step } j \text{ on module } M_k \text{ placed} \\ & \text{with its top-left corner over cell } c_l \\ 0, & \text{otherwise} \end{cases}$$

Such a variable captures the allocation and binding (operation O_i is executing on module M_k), the scheduling (O_i starts to execute at time-step j, with a duration of $C_i^{M_k}$) and the placement (the top-left corner of module M_k is placed over cell c_l). For example, considering the dilution operation implemented as in Figure 2(c), the binary variable will be expressed as:

$$z_{i,j,k,l} = \begin{cases} 1, & \text{if i=7, j=1, k=Diluter, l=46} \\ 0, & \text{otherwise} \end{cases}$$

By using the defined variable, the start time of an operation $O_i \in \mathcal{V}$ becomes:

$$t_i^{start} = \sum_j \sum_k \sum_l j \times z_{i,j,k,l}, \quad \forall O_i \in \mathcal{V}, \tag{2}$$

where j represents the time-step when the operation starts executing.

A. Scheduling and precedence constraints

The scheduling constraint requires that every operation O_i be scheduled only once:

$$\sum_j \sum_k \sum_l z_{i,j,k,l} = 1, \quad \forall O_i \in \mathcal{V}. \tag{3}$$

For each edge in the application graph we have to introduce a precedence constraint. Consider the operations O_i and $O_n \in \mathcal{V}$ for which there exists a dependency $e_{i,n} \in \mathcal{E}$ in the sequencing graph $\mathcal{G}$. Then O_n must be scheduled for execution only after the completion of O_i:

$$t_i^{start} + \sum_j \sum_k \sum_l \left(C_i^{M_k} \times z_{i,j,k,l} \right) \leq t_n^{start}, \quad \forall O_i \text{ and } O_n \text{ such that } \exists e_{i,n} \in \mathcal{E}. \tag{4}$$

For example, considering operations O_9 and O_{10} in Figure 2(a), with O_{10} depending on O_9, we have $t_9^{finish} \leq t_{10}^{start}$. If O_n is not scheduled immediately after the completion of O_i then a storage module is required. The number of such storage modules during a time-step j is important in defining the placement constraints for the model, since the storage modules also occupy chip area. Using a binary variable $m_{i,j}$ defined as:

$$m_{i,j} = \begin{cases} 1, & \text{if a storage unit is needed for } O_i \text{ in step } j \\ 0, & \text{otherwise} \end{cases}$$

we can capture the number of storage units required during a time-step j. Thus, at time-step j, the binary variable associated with the edge between operations O_i and O_n is expressed as:

$$\sum_{h=1}^{j-C_i^{M_k}} \sum_k \sum_l z_{i,h,k,l} - \sum_{h=1}^{j} \sum_k \sum_l z_{n,h,k,l} = m_{i,j}, \quad \forall j \in \mathcal{T}, \forall O_i, O_n \in \mathcal{V} \text{ such that } \exists e_{i,n} \in \mathcal{E} \tag{5}$$

Variable $m_{i,j}$ will have the value 1 at that time-step j when O_i has finished executing (first sum of the equation equals 1), but O_j has not started yet (second term of the equation is 0).

B. Resource constraints

Considering the fact that two operations of the same type can be bound to the same resource, a constraint must be expressed to prevent the overlapping of these operations during their execution. An operation O_i is executing at time-step j if:

$$\sum_{h=j-C_i^{M_k}+1}^{j} \sum_k \sum_l z_{i,h,k,l} = 1, \quad \forall O_i \in \mathcal{V}.$$

Thus, at any time-step $j \in \mathcal{T}$ and for any module $M_k \in \mathcal{L}$ there must be at most one operation that is executing:

$$\sum_i \sum_{h=j-C_i^{M_k}+1}^{j} \sum_l z_{i,h,k,l} \leq 1., \quad \forall M_k \in \mathcal{L}, j \in \mathcal{T}. \tag{6}$$

C. Placement constraints

The allocated modules have to be placed on-chip such that they do not *physically* overlap. However, since a biochip is reconfigurable, the same cell area can be used by two different modules as long as they do not overlap *in time*. Hence, the placement constraints will be expressed as a function of time, considering each time-step j in the schedule.

The first constraint to be considered is the size of the microfluidic array of the biochip. At each time step j, the sum of the modules that are placed on the array should not exceed the total area size, $m \times n$:

$$\sum_i \sum_{h=j-C_i^{M_k}+1}^{j} \sum_k \sum_l z_{i,h,k,l} \times L_k \times W_k \leq m \times n, \forall j \in \mathcal{T} \tag{7}$$

where L_k and W_k are the length and width of module M_k, respectively, measured in number of cells.

The second constraint captures that no modules should overlap, i.e., a cell c_l on the array can be occupied by at most one module during time step t_j.

Let us consider a cell c_r (with coordinates x_r and y_r) which is the top-left corner of module M_k. If cell c_l is within the rectangle formed by M_k, i.e., $x_l - L_k + 1 \leq x_r \leq x_l$ and $y_l - W_k + 1 \leq y_r \leq y_l$, then we have to impose the restriction that no other module is active during this time interval:

$$\sum_i \sum_{h=j-C_i^{M_k}+1}^{k} \sum_k \sum_r z_{i,h,k,r} \leq 1. \tag{8}$$

D. Local Branching

Given the ILP model defined above, an ILP solver will search the design space extensively to produce a *provable* optimal solution. However, for practical applications, we are often interested to produce good quality solutions in a reasonable time. Local Branching (LB) [14] is a meta-heuristic for design space exploration that controls the behavior of the ILP solver, by directing it to quickly explore well-defined local neighborhoods. LB is applied within the framework of ILP, by automatically introducing search-limiting constraints into the ILP model, and interacting with the ILP solver.

Figure 3 illustrates how LB works. The search starts from an initial feasible solution $\overline{x}$. Next, we define the k-OPT neighborhood $\mathcal{N}(\overline{x},k)$ of $\overline{x}$, where k is a parameter that determines

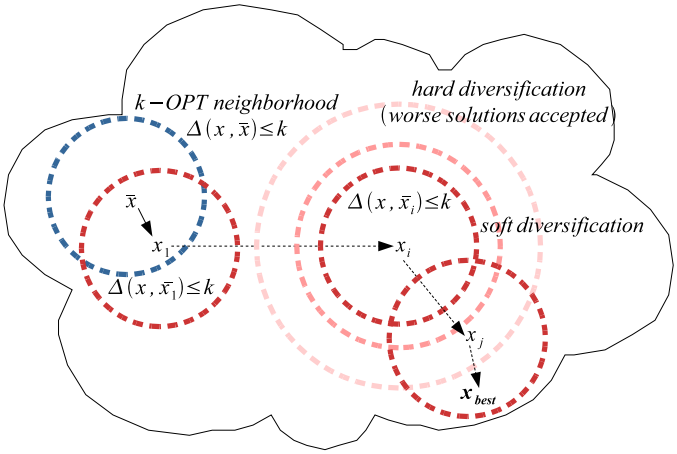


Fig. 3: Exploring the design space using Local Branching

the neighborhood size. Those solutions x are included in $\mathcal{N}$, which satisfy the following constraint:

$$\Delta(x,\overline{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k, \tag{9}$$

where $\mathcal{B}$ is the set of binary variables that define the solution, $x_j \in \mathcal{B}$ is a binary variable, and $\overline{S}$ is the subset of $\mathcal{B}$ for which $x_j = 1$. Thus, Δ will count the number of binary variables in x that change their value (from 1 to 0 or viceversa), with respect to $\overline{x}$.

By varying k, we can control the size of $\mathcal{N}$. We would like a size that is much easier to solve than the entire design space, but still large enough to contain better solutions than $\overline{x}$. To speed up the search, LB can apply time limits to the exploration of any local neighborhood. Once a better solution x_1 has been found, the search continues in a similar manner, by defining a neighborhood $\mathcal{N}(\overline{x}_1,k)$, such that $\Delta(x,\overline{x}_1) \leq k$, see Figure 3. If we are not able to find a better solution starting from an x_i, LB uses *diversification*.

Soft diversification enlarges the local neighborhood by slightly increasing the value of k, in the hope of finding a better solution. In case such a solution is not found, LB can employ *strong* diversification, where not only that k is further enlarged, but worse solutions than the current one are accepted, to guide the search into unexplored areas. LB stops when an imposed time limit has been reached, or when the number of allowed diversifications has been exceeded. The returned result is the best solution x_{best} found so far.

V. Experimental Evaluation

We were interested to evaluate the ILP-based approach proposed in the previous section. For this purpose, we have used two real-life examples: (1) *In-vitro* diagnostics on human physiological fluids (IVD) [8]. IVD has 15 operations. (2) The mixing stage of a polymerase chain reaction application (PCR/M) [7], which is one of the most common techniques for DNA analysis. PCR/M has 5 operations. We have solved the ILP model with GAMS 21.5 using the CPLEX 9.130 solver, running on Sun Fire v440 computers with 4 UltraSPARC IIIi CPUs at 1,062 MHz and 8 GB of RAM. The results are presented in Table I.

TABLE I: Comparison of SF, OS and LB approaches

App.	Area	SF	OS	Exec. Time	LB	Exec. Time
IVD	10x10	14 s	14 s	70 min	14 s	6 min
IVD	8x8	23 s	19 s	43 min	21 s	20 min
IVD	6x6	49 s	38 s	30 min	38 s	10 min
PCR	10x10	17 s	13 s	37 min	13 s	1 min
PCR	8x8	19 s	15 s	19 min	15 s	6 min
PCR	6x6	38 s	28 s	15 min	34 s	10 min

For each application, we have considered the library from Figure 1(c) and three, progressively smaller, area constraints (second column of Table I). We have performed the allocation, binding, scheduling and placement such that the application completion time is minimized[1]. We used three approaches to derive the implementations:

1) The *straight-forward* approach (SF) does architectural synthesis (allocation, binding and scheduling) separately from placement. First, an implementation Ψ^0 is derived using our ILP model, limited by the total chip area, but without the placement constraints. Next, we attempt the placement of Ψ^0 on the available area, modifying the scheduling if required to fit the modules, thus obtaining the final implementation Ψ. For both of these steps we have obtained the optimal solutions. The schedule length of Ψ is presented in column 4.
2) The *Optimal Synthesis* approach (OS) outlined in the previous section. The schedule lengths and runtime required by the CPLEX solver to produce the optimal solutions are presented in columns 4 and 5, respectively.
3) The *Local Branching* (LB) approach, which quickly guides the ILP solver to good quality solutions. The schedule lengths obtained with LB and the *time limits* imposed by us on the LB algorithm are presented in the last two columns of the table.

As we can see from Table I, considering placement at the same time with architectural synthesis (OS) can lead to significant improvements over SF, which does not take into account placement. On average, we have obtained an 18.2% improvement on the bio-application completion time, with up to 26.3% improvement for the PCR application on a 6x6 array. We can see that considering the placement is especially important as the biochip area is reduced. Our OS approach has been able to find the optimal solutions in a reasonable time, for example, the IVD is synthesized optimally on an 8x8 array in 43 minutes.

However, biochemical applications are becoming increasingly complex, and thus scalable methods will be required for their synthesis. Hence, we have proposed the LB approach which can produce good quality solutions in a reasonable time. For the applications in Table I, LB has been able to obtain the optimal solutions for most of the cases, in a fraction of the time needed by OS, with an average loss of quality of only 5.3%. Such an approach is useful for larger applications which are intractable by OS.

[1]The detection operations were ignored.

VI. Conclusion

In this paper we have addressed the synthesis of microfluidic-based biochips, which are based on the manipulation of liquids not as a continuous flow, but as discrete droplets, and hence are highly reconfigurable and scalable. We have modeled a biochemical application using a acyclic polar graph, where each node is an operation and the edges represent dependencies between the operations.

We have considered architectural synthesis at the same time with physical synthesis: we have proposed an ILP-based synthesis methodology for the unified allocation, placement, binding and scheduling of operations on the biochip. The ILP model has been complemented with a search heuristic that can quickly guide the solver to good quality solutions.

Using two real-life examples, we have shown that our ILP-based approach can successfully synthesize the application and find the optimal completion time, under given area constraints. As the experimental section shows, considering the placement during architectural synthesis leads to significantly better quality solutions.

References

[1] T. Thorsen, S. Maerkl, and S. Quake, "Microfluidic largescale integration," *Sci.*, vol. 298, pp. 580–584, 2002.
[2] E. Verpoorte and N. F. D. Rooij, "Microfluidics meets mems," *Proc. IEEE*, vol. 91, pp. 930–953, 2003.
[3] M. G. Pollack, A. D. Shenderov, and R. B. Fair, "Electrowetting-based actuation of droplets for integrated microfluidics," *Lab Chip J.*, vol. 2, pp. 96–101, 2002.
[4] T. Zhang, K. Chakrabarty, and R. B. Fair, *Microelectrofluidic Systems: Modeling and Simulation*. Boca Raton, FL: CRC Press, 2002.
[5] K. Chakrabarty and J. Zeng, "Design automation for microfluidics-based biochips," *ACM J. on Emerging Technologies in Comput. Syst.*, vol. 1, no. 3, pp. 186–223, 2005.
[6] F. Su and K. Chakrabarty, "Architectural-level synthesis of digital microfluidics-based biochips," in *Proc. Int. Conf. Comput. Aided Des.*, 2004, pp. 223–228.
[7] ——, "Module placement for fault-tolerant microfluidics-based biochips," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 11, no. 3, pp. 682–710, 2006.
[8] F. Su, W. Hwang, and K. Chakrabarty, "Droplet routing in the synthesis of digital microfluidic biochips," in *Proc. Des., Automat. and Test in Europe Conf.*, vol. 1, 2006, pp. 73–78.
[9] M. Cho and D. Z. Pan, "A high-performance droplet router for digital microfluidic biochips," in *Proc. Int. Symp. Phys. Des. (in press)*, 2008.
[10] F. Su and K. Chakrabarty, "Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips," in *Proceedings of the 42nd annual conference on Design automation*. New York, NY, USA: ACM, 2005, pp. 825–830.
[11] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Placement of digital microfluidic biochips using the t-tree formulation," in *Proc. Design Automation Conference*, 2006, pp. 931–934.
[12] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Des. Test*, vol. 17, no. 1, pp. 68–83, 2000.
[13] A. A. E. Farag, H. M. El-Boghdadi, and S. I. Shaheen, "Improving utilization of reconfigurable resources using two-dimensional compaction," *J. Supercomput.*, vol. 42, no. 2, pp. 235–250, 2007.
[14] M. Fischetti and A. Lodi, "Local branching," *Mathematical Programming*, vol. 98, no. 1-3, pp. 23–47, 2003.
[15] R. B. Fair, "Digital microfluidics: is a true lab-on-a-chip possible?" *Microfluidics and Nanofluidics*, vol. 3, no. 3, pp. 245–281, 2007.
[16] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Science, 1994.
[17] D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, pp. 384–393, 1975.