DCSA: Distributed Channel-Storage Architecture for Flow-Based Microfluidic Biochips

Chunfeng Liu, Xing Huang, Bing Li, Hailong Yao, Paul Pop, Tsung-Yi Ho and Ulf Schlichtmann

***Abstract*—Flow-based microfluidic biochips have attracted much attention in the EDA community due to their miniaturized size and execution efficiency. Previous research, however, still follows the traditional computing model with a dedicated storage unit, which actually becomes a bottleneck of the performance of biochips. In this paper, we propose a distributed channel-storage architecture (DCSA) to cache fluid samples inside flow channels temporarily. Since distributed storage can be accessed more efficiently than a dedicated storage unit and channels can switch between the roles of transportation and storage easily, biochips with this architecture can achieve a higher execution efficiency even with fewer resources. Furthermore, we also address the flow-path planning that enables the manipulation of actual fluid transportation/caching on a chip. Simulation results confirm that the execution efficiency of a bioassay can be improved significantly, while the number of valves in the biochip can be reduced accordingly. Also, flow paths for transportation tasks can be constructed and planned automatically with minimum extra resources.**

***Index Terms*—Microfluidic biochips, channel storage, scheduling, architectural synthesis, flow-path mapping.**

I. Introduction

The emergence of microfluidic biochips has reshaped the traditional biochemical procedures due to their high execution efficiency and miniaturized fluid manipulation [2], [3]. With this miniaturization, biochemical assays can be scaled down to nanoliter volumes, so that precious reagents can be saved to reduce the cost of experiments. Since operations executed on a biochip are automated and controlled by a microcontroller, the reliability in executing biochemical experiments can also be improved significantly compared with the traditional manual-based experimental flow [4]. Accordingly, this lab-on-a-chip device is now increasingly being used in many areas of biochemistry and biomedical applications, such as immunoassays [5], drug discovery [6], and DNA analysis [7].

Microfluidic biochips based on continuous flow use valves to control the movement of samples and reagents. The structure of a biochip is illustrated in Fig. 1(a) [8]. On top of the glass substrate, a flow channel is constructed for the transportation of fluids. Above the flow channel, a control channel connected to an air pressure source is used to control the open/closed state of the valve. Both channels are built from elastic materials, so that air pressure in the control channel squeezes the flow channel below to block the movement of the fluid. Conversely, if the pressure in the control channel is released, the fluid can resume its movement.

With valves as the basic controlling components, complex devices can be constructed [9]. For example, the structure of a mixer is shown in Fig. 1(b) [10]. The three valves at the top form a peristaltic pump by switching alternately at a sufficiently high frequency to drive the mixing of fluids. Besides mixers, a dedicated storage unit can also be built from channels and valves. Fig. 1(c) shows the schematic of a biochip with a mixer connected to a storage unit [11], where eight side-by-side cells are used to store fluid samples. Furthermore, at a port of the storage unit, valves in a multiplexer-like structure direct a fluid sample to enter or leave a specific cell. This concept of dedicated storage, while easy to operate, suffers however from several other limitations, including constrained capacity of the storage unit, limited access bandwidth at input/output ports, fixed position on the chip, and large area occupation [12].

Biochips are used to execute operations in biochemical assays, whose protocols are usually described by a static, directed, and acyclic graph called sequencing graph. In Fig. 2(a), the sequencing graph of the mixing phase of the polymerase chain reaction (PCR) is illustrated [13]. This assay takes eight input samples (i_1-i_8) and processes them with seven mixing operations (o_1-o_7) to generate copies of a DNA sequence. Moreover, the edges in the sequencing graph define the dependency of operations, where a parent operation should always be executed before its child operations.

To design a high-efficiency chip architecture for bioassay execution, the traditional design flow is usually divided into two phases [14]. In the first stage, biochemical operations are bound to specific devices and scheduled to execute according to the dependencies specified by the sequencing graph. The target of this step is to find a resource binding and scheduling solution such that the total execution time of the bioassay is minimized. Then, in the second stage, physical design is per-

A preliminary version of this paper was published in the Proceedings of the 54th Annual Design Automation Conference (DAC), 2017 [1]. Major extensions beyond [1] include a detailed analysis of the design challenges involved in flow-path planning, an updated problem formulation considering resource binding and scheduling, architectural synthesis, and flow-path planning simultaneously, an effective ILP-based flow-path planning method, a deadlock removal strategy, two transportation-conflict elimination techniques, as well as comprehensive experimental evaluations.

Chunfeng Liu, Bing Li, and Ulf Schlichtmann are with the Chair of Electronic Design Automation, Technical University of Munich, Germany (e-mail: chunfeng.liu@tum.de; b.li@tum.de; ulf.schlichtmann@tum.de).

Xing Huang is with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China (e-mail: xing.huang1010@gmail.com).

Xing Huang and Tsung-Yi Ho are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (e-mail: xing.huang1010@gmail.com, tyho@cs.nthu.edu.tw).

Hailong Yao is with the Department of Computer Science and Technology, Tsinghua University, Beijing, China (e-mail: hailongyao@tsinghua.edu.cn).

Paul Pop is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark (e-mail: paupo@dtu.dk).

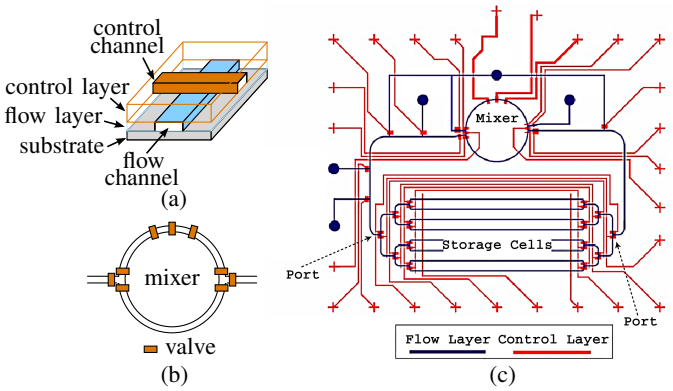

Fig. 1: Components and structure of flow-based biochips. (a) Valve structure. (b) Mixer. (c) A biochip with eight storage cells [11].

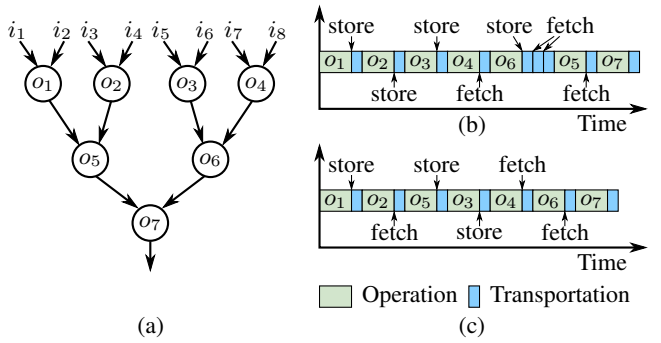

Fig. 2: Sequencing graph and different schemes of scheduling with one mixer. (a) Sequencing graph of PCR. (b) Scheduling with four store operations. Required storage capacity is three. (c) Scheduling with three store operations. Required storage capacity is two.

formed to assign exact locations for the allocated devices and construct flow channels to connect them, thereby generating a biochip architecture with minimized total cost.

With a certain biochip architecture, when executing a bioassay, each fluid transportation task needs to be mapped to one or more continuous flow-channels to form a complete flow path between devices. Moreover, to drive the movement of fluid sample and recover the corresponding waste liquid, a pressure pump and a waste port need to be placed at the start and end of the flow path, respectively. More importantly, all these flow paths as well as the placement positions of corresponding pressure pumps and waste ports should be considered systematically, so that all the transportation tasks can be performed without any conflict, while the extra resources (e.g., pumps, ports, etc.) introduced to the biochip are minimized.

Over the past few years, a number of methodologies have been proposed to deal with the design challenges of biochips. The method in [14] proposes a top-down flow to generate a biochip architecture while minimizing the execution time of the assay. The methods in [15] and [16] propose to solve the flow-channel routing problem considering obstacles, while placement/routing iterations are performed in [17] to reduce flow-channel crossings. Control logic synthesis is investigated in [18] to reduce the number of control ports, in [19] to optimize valve switching considering the relation between control patterns, and in [20] to match lengths of control channels. The method in [21] proposes to solve the control-port minimization problem in both high-level synthesis and physical design stages. In [22], flow layer and control layer codesign is proposed to achieve valid routing on both layers iteratively, and in [23] interactions on both flow and control layers are modeled by an integer linear programming (ILP) formulation to achieve a better codesign efficiency. All the aforementioned design methods are, however, still based on the traditional dedicated storage architecture. The vast flexibility of flow channels in both fluid transportation and caching has so far been ignored.

In this paper, we propose the first synthesis framework that models distributed channel storage and time multiplexing at intersections of flow channels to execute bioassays efficiently. Our contributions are summarized as follows:

- Instead of using a dedicated storage unit, we cache intermediate fluid samples in flow channels with time multiplexing at the intersections of channels, so that not only the access bandwidth limit at the ports of the dedicated storage unit is overcome, but also the efficiency of channels and valves is improved.
- This is the first work to consider storage minimization from scheduling to architectural synthesis, so that the execution time of the assay can be reduced effectively.
- The problem of flow-path planning on a certain biochip architecture with distributed channel storage is addressed for the first time. With the proposed method, the manipulation of actual fluid transportation/caching can be realized without any conflict. Meanwhile, extra resources introduced to the biochip can also be minimized.
- Two transportation-conflict elimination techniques, including a scheduling adjustment technique and an architecture adjustment technique, are proposed to fundamentally address the resource contention problem in flow-path planning. Moreover, a deadlock-removal strategy is proposed to effectively coordinate the execution of both fluid transportation and caching.
- Simulation results on multiple benchmarks confirm that the proposed framework leads to shorter execution time of bioassays, lower total cost of biochips, and higher efficiency of resource utilization.

The rest of this paper is organized as follows. In Section II, we explain the motivation of storage synthesis and analyze the design challenges involved in flow-path planning, thereby formulating the problem we address in this paper. In Section III, we describe the techniques to reduce storage usage and to model distributed channel storage in scheduling and architectural synthesis. In Section IV, the proposed method for solving the flow-path planning problem is described in detail. Section V presents the simulation results. Finally, the conclusions are drawn in Section VI.

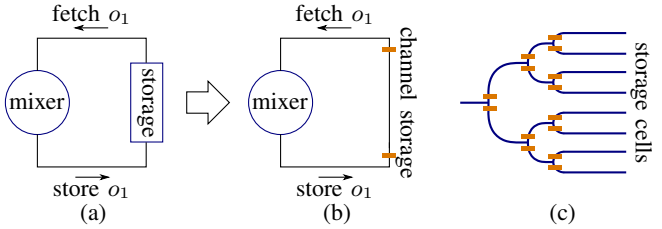

Fig. 3: Storage mechanisms. (a) Storage with dedicated unit. (b) Channel storage. (c) Port of dedicated storage unit.

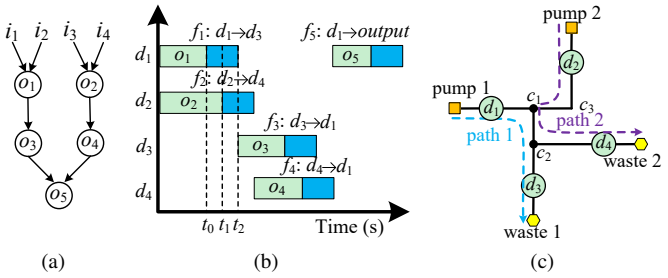

Fig. 4: Flow-path planning. (a) Sequencing graph of a bioassay. (b) A scheduling and binding solution with minimized execution time. (c) Flow paths constructed on a given chip architecture.

II. Motivation and Problem Formulation

A. Storage Minimization and Proposed Concept of Distributed Channel Storage

The sequencing graph of an assay defines the dependency of operations. These operations are scheduled to devices in a given order for execution. Different schedules, however, yield different storage usage and transportation requirements. In Fig. 2(b) and 2(c), two schedules for the PCR assay are shown, where one mixer is used to execute the operations. The first schedule executes the operations in the order $o_1 \rightarrow o_2 \rightarrow o_3 \rightarrow o_4 \rightarrow o_6 \rightarrow o_5 \rightarrow o_7$. After executing o_1, the intermediate result should be transported to the storage unit, so that the device can be reused to execute o_2. When o_6 is executed, it takes the result of o_4 from the mixer directly as one input and fetches the result of o_3 from the storage unit. In this schedule, in total four storage operations and four fetch operations are needed. In addition, the results of o_1, o_2 and o_3 stay in the storage unit at the same time, so that the capacity requirement of the storage unit is three. In the schedule in Fig. 2(c) with the execution order $o_1 \rightarrow o_2 \rightarrow o_5 \rightarrow o_3 \rightarrow o_4 \rightarrow o_6 \rightarrow o_7$, however, there are only three store and three fetch operations, leading to a storage capacity of only two units. In addition, the execution time of the assay in the second schedule is even shorter.

The comparison of the two schedules in Fig. 2(b) and 2(c) demonstrates that the schedule scheme affects the transportation of fluid samples as well as the required capacity of the storage unit, and the execution time of the assay may be unnecessarily prolonged if storage and transportation are not considered properly. This important problem, however, has not been dealt with by previous methods.

When a fluid sample is stored, it is transported to the storage unit through a channel. The diagram of a simple chip with one mixer and one storage unit is shown in Fig. 3(a). When this chip is used to execute the operation $o_1 \rightarrow o_2 \rightarrow o_5$ in the schedule in Fig. 2(c), the mixer first stores the result of o_1 in the storage unit. After o_2 is finished, the result of o_1 is fetched back to the mixer to execute the operation o_5. In this case, the channel itself instead of a dedicated storage unit is sufficient to store the intermediate result from o_1, as shown in Fig. 3(b). Note that valves need to be placed at the two ends of the channel, so that the fluid can be cached in the channel safely. This example demonstrates that fluid samples can in fact be cached within temporary storage constructed from channel segments. This distributed storage can also overcome the bandwidth limit problem at the ports of the dedicated storage unit illustrated in Fig. 1(c) and Fig. 3(c), where multiple fluid samples must be queued when they access the storage unit simultaneously.

B. Design Challenges in Flow-Path Planning

When executing bioassays on a microfluidic biochip, fluid samples need to be transported to specific devices/channel segments at certain time windows according to the scheduling and binding solution and this, undoubtedly, requires a carefully planned flow-channel network to complete these tasks. For a transportation task between two devices denoted as d_i and d_j, to drive the movement of fluid, an external pressure source or a mechanical pump needs to be connected to d_i to produce thrust. Moreover, to avoid channel blockage after the completion of the corresponding operation, waste liquid that is not needed anymore should be discarded, thus requiring a waste port connected to d_j. As a consequence, the pump, the two devices, the waste port, as well as the channel segments connecting them form the complete flow path of the transportation task.

We take the sequencing graph shown in Fig. 4(a) as an example to illustrate the flow-path planning on a certain biochip. Fig. 4(b) shows a scheduling and binding solution with minimized execution time of the bioassay described in Fig. 4(a), where four devices (d_1–d_4) are allocated and five transportation tasks (f_1–f_5) are scheduled to execute corresponding operations. For example, in task f_1, the resulting fluid of operation o_1 is required to be transported to device d_3 as an input sample of operation o_3, leading to a flow path from d_1 to d_3. Fig. 4(c) shows a given biochip architecture, on which $path$ 1 is a feasible flow path starting with a mechanical pump, passing through devices d_1 and d_3, and ending at a waste port. Note that a valid flow path should bypass those devices occupied by other on-going operations as well as flow channels used for fluid caching.

1) Transportation conflict: Since several transportation tasks may be performed concurrently in a scheduling scheme, to avoid potential transportation conflicts, flow paths among these tasks need to be considered systematically. Let us revisit the scheduling scheme in Fig. 4(b). Since the time windows for executing transportation tasks f_1 and f_2 overlap with each other, the corresponding flow paths cannot share any on-chip

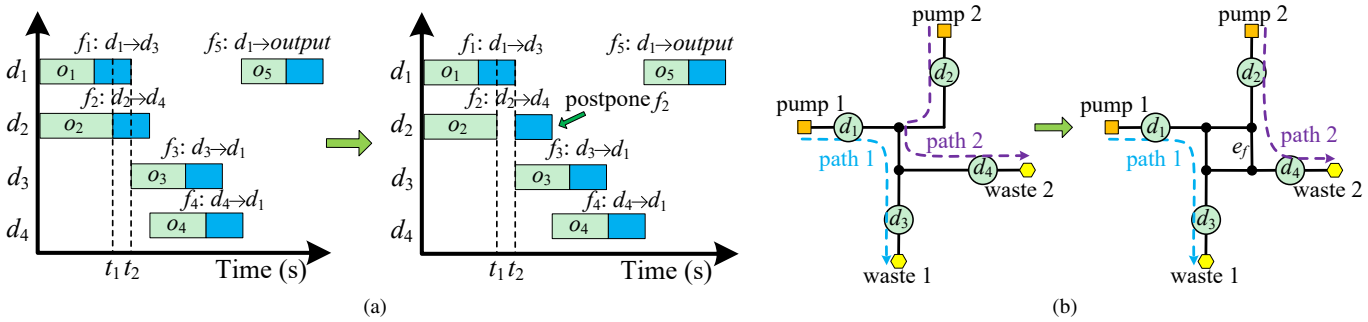

Fig. 5: Strategies for transportation conflict elimination. (a) Scheduling adjustment. (b) Architecture ajustment.

resource, including pump ports, channel segments, etc. For example, *path* 1 and *path* 2 in Fig. 4(c) are flow paths constructed for tasks f_1 and f_2, respectively. These two paths, however, are essentially in transportation conflict with each other, since switches c_1 and c_2 as well as the channel segment between them are shared by both paths. Worse still, we cannot even find a valid path-planning solution for the two tasks due to the limited on-chip resources in Fig. 4(c).

Accordingly, we propose two conflict elimination techniques, *scheduling adjustment* and *architecture adjustment* as illustrated in Fig. 5, to fundamentally solve the transportation conflict discussed above.

Scheduling adjustment: The goal of scheduling adjustment is to eliminate transportation conflicts in a flow-path planning solution by postponing the execution of some transportation tasks. As illustrated in Fig. 5(a), to eliminate the conflict between the flow paths described in Fig. 4(c), the execution of transportation task f_2 is postponed by the scheduling adjustment operation, so that tasks f_1 and f_2 do not overlap in their execution time. As a result, the shared channel segment between switches c_1 and c_2 in Fig. 4(c) can be used by both paths sequentially, although the completion time of the bioassay is increased slightly.

Architecture adjustment: Another strategy to eliminate the transportation conflicts is to introduce extra resources to the biochip architecture directly, so that the previously shared on-chip resources can be released from conflicts. For example, in Fig. 5(b), a channel segment e_f with two extra switches is added to the chip architecture, leading to a new flow path for transportation task f_2. Since the new path does not share any resource with that of task f_1, the previous conflict is removed from the chip successfully.

2) Caching deadlock: Another issue that needs to be considered during flow-path planning is caching deadlock. Since flow channels in a distributed channel-storage architecture fulfill the dual functions of transportation and caching, channel segments used for fluid caching may block the normal transportation of other fluids, thereby leading a deadlock of the flow-channel network. For example, when mapping the scheduling scheme described in Fig. 6(a) to the biochip architecture shown in Fig. 6(b), the resulting fluids of operations o_1 and o_2 need to be transported from devices d_1 and d_2 to d_3, respectively. To wait for the completion of operation o_3, whose resulting fluid is another input of operation o_4, the

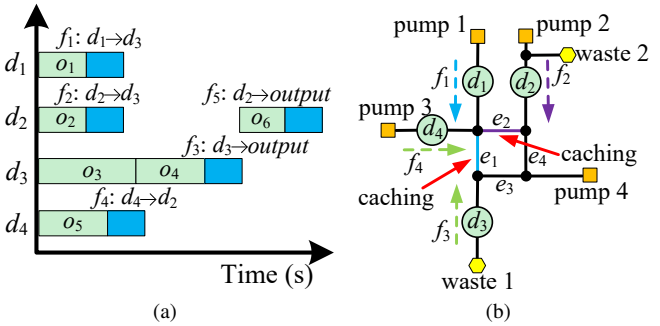

Fig. 6: Illustration of caching deadlock. (a) Scheduling scheme of a bioassay. (b) A flow-path planning solution with caching deadlock.

two fluids are temporally cached in channel segments e_1 and e_2, thus blocking the flow path of transportation task f_4. To solve this problem, one feasible solution is to desynchronize the execution of these transportation tasks by employing an idea similar to the proposed scheduling adjustment strategy. For example, if the execution of f_4 is postponed to a point in time that the fluid cached in either e_1 or e_2 has been removed, the deadlock in Fig. 6(b) can be eliminated. Another way to solve this problem is to release these blocked channels by directly moving the corresponding cached fluids to other unused channel segments. For example, if the resulting fluid of o_2 is moved to either channel segment e_3 or e_4 for caching, the deadlock can also be eliminated.

C. Problem Formulation

Based on the proposed concept of distributed channel storage, the synthesis problem considering architecture design and path planning in flow-based microfluidic biochips can be formulated as follows:

Inputs: The sequencing graph of a biochemical assay; the execution durations of all operations; the maximum numbers of devices allowed in the chip.

Outputs: A schedule minimizing intermediate fluid storage; a channel caching schedule including the locations and the time slots of fluid samples stored temporarily in channels; a biochip architecture with the exact flow paths to manipulate the transportation/caching of fluid samples without conflict.

Objectives: Minimizing the overall resource usage and the completion time of the bioassay.

TABLE I: Variables and Constraints for Scheduling and Binding

Variables:	
O:	set of nodes (operations) in the sequencing graph;
o_i:	operation indexed by i;
t_i^s:	starting time of operation o_i;
t_i^e:	ending time of operation o_i;
u_i:	duration of operation o_i;
E:	set of edges in the sequencing graph;
(o_i, o_j):	edge from o_i to o_j where o_i is the parent of o_j;
$u_{i,j}$:	transportation/storage time from o_i to o_j.
D:	set of devices in the biochip;
d_i:	device indexed by i;
$s_{i,k}$:	a 0-1 variable representing whether o_i is assigned to d_k.
Constraints:	
Uniqueness:	$\sum_{k=1}^{\lvert D\rvert} s_{i,k} = 1, \quad \forall o_i \in O$ (1)
Duration:	$t_i^s + u_i \le t_i^e, \quad \forall o_i \in O$ (2)
Precedence:	$t_i^e + u_{i,j} \le t_j^s, \quad \forall (o_i, o_j) \in E$ (3)
Non-overlapping:	$s_{i,k} + s_{j,k} \le 1 \quad \texttt{if } t_i^s < t_j^e \bigwedge t_i^e > t_j^s, \quad \forall o_i, o_j \in O,\ d_k \in D$ (4)

III. Synthesis of Biochips Considering Storage and Caching

Storage and caching should be considered from scheduling to architectural synthesis to reduce storage requirements and to construct channels for distributed storage. Correspondingly, the proposed method includes three major parts: 1) intermediate fluid storage is minimized in scheduling and binding, 2) all the allocated devices are assigned to exact locations on a biochip, and 3) channel segments that cache intermediate fluid samples are constructed among devices, leading to a distributed channel-storage system which fulfills the tasks of transportation and storage at the same time.

A. Minimizing Storage in Scheduling and Binding

Scheduling and binding assign operations in a given sequencing graph to time slots of specific devices. To optimize storage requirements, we formulate the scheduling and binding task as an ILP problem.

The variables and constraints for scheduling and binding are listed in Table I. The uniqueness constraint (1) specifies that operation o_i should be assigned to one device only. The duration constraint ensures o_i has enough time to finish. The precedence constraint (3) guarantees that a child operation must be executed later than its parents. Finally, the non-overlapping constraint (4) prevents two operations whose operation periods overlap from being executed by the same device. These constraints are common for high-level synthesis and widely used in synthesis methods for biochips [24].

To minimize the execution time of the assay, another variable t^E is used to represent the latest ending time of all operations, constrained as

$$t_i^e \le t^E, \quad \forall o_i \in O. \tag{5}$$

By minimizing t^E, the operations in the sequencing graph are assigned to proper time slots to produce a compact schedule.

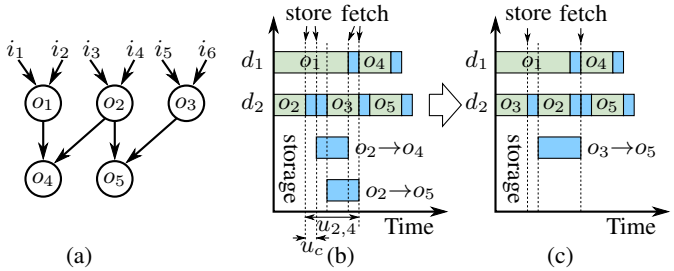

Fig. 7: Storage reduction. (a) Sequencing graph. (b) Schedule with two storage requirements. (c) Schedule with one storage requirement. The execution times of the assay with these two schedules are equal.

To reduce storage requirements, we introduce an additional *storage minimization objective*. Assume that the pure transportation time from a device to another device is a constant u_c. If the schedule produces a transportation time larger than u_c, the fluid sample must be cached somewhere before it is used (u_c is set to 1 in our experiments). Fig. 7 illustrates the schedules of executing an assay with five operations by two devices. The sequencing graph of this assay is shown in Fig. 7(a). In Fig. 7(b), operation o_2 is scheduled before o_3. Consequently, the result of o_2 must be stored until it is used by o_4 and o_5, leading to two storage requirements. In Fig. 7(c), o_3 is scheduled before o_2, leading to only one storage requirement lasting a shorter time. In this example, we can observe that the lifetime of stored fluid samples is determined by the difference between the ending time of the parent operation and the starting time of the child operation $u_{i,j}$ defined in Table I. Consequently, the total storage requirement can be reduced by solving the following ILP problem:

$$\text{minimize} \quad \alpha t^E + \beta \sum_{(o_i,o_j)\in E \bigwedge d_i \neq d_j} u_{i,j} \tag{6}$$

$$\text{subject to} \quad (1)\text{–}(5) \tag{7}$$

where α and β are constants to control the priority of execution time and storage requirement minimization. $d_i \neq d_j$ excludes the operation pairs assigned to the same device so that they do not need transportation.

B. Architectural Synthesis with Distributed Channel Storage

After solving the optimization problem (6)–(7), we have a schedule similar to Fig. 7(c), including the information: 1) to which devices operations are assigned; 2) the starting time and the ending time of each operation; 3) the starting time and the ending time of each fluid storage requirement.

The schedule, however, only defines the transportation requirements between devices after operations are executed. In the chip, physical channels need be constructed to conduct these transportation tasks. When a large assay is executed by several devices, transportation channels need to be built between nearly any pair of devices to move fluid samples efficiently. Since the flow-layer in a biochip is only two-dimensional, eventually intersections between channels cannot be avoided. At an intersection, a switch should be built to

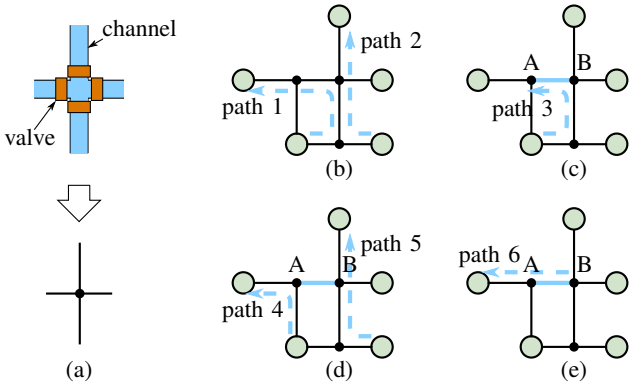

Fig. 8: Switch and channel storage. Large nodes represent devices and small nodes represent switches. (a) Switch structure. (b) Two paths sharing one channel segment with time multiplexing. (c) Fluid sample to channel storage. (d) Storage in channel segment. (d) Fluid sample from channel storage to device.

direct the transportation flow to the target device. A switch is constructed by four valves at an intersection, as shown in Fig. 8(a). At a given moment, two out of the four valves in a switch are opened to connect two channel segments. Consequently, a transportation channel between two devices becomes a path formed by several channel segments connected by switches. Such a path is called a *transportation path* henceforth.

Besides channels, the locations of devices should also be determined. These locations should be assigned together with the construction of transportation channels, because the distance and relative locations of devices affect how channels are constructed and how they intersect with each other.

Considering devices and channels together, the architecture of a biochip can be described as devices surrounded by channel segments in the form of a grid. For example, the architecture of a biochip with five devices is shown in Fig. 8(b), where the smaller nodes represent switches and the larger nodes represent devices. Transportation paths between devices are formed from channel segments connected by switches, e.g., path 1 and path 2 in Fig. 8(b). Since transportation paths are used only when there is a fluid sample traveling along it, channel segments can be reused by different paths so that the efficiency of channel segments increases.

With transportation paths formed from channel segments, the proposed distributed channel storage concept can thus be formulated. As illustrated in Fig. 3(c), a dedicated storage unit suffers bandwidth problem at its ports because multiple fluid samples must be queued when they access the storage unit at the same time. Observing that the storage cells inside a dedicated storage unit are in fact channel segments, as shown in Fig. 1(c), we distribute fluid storage directly in channel segments. For example, in Fig. 8(c), along path 3 a fluid sample is moved to the channel segment between A and B. However, the next operation using this fluid sample is scheduled later, so that the fluid sample must stay in the channel segment. During the lifetime of this storage, the channel segment between A and B cannot be used by other paths and the valves at the two ends of this channel segment should be closed. Further transportation tasks between devices are, however, still fulfilled by paths not including this channel segment, as path 4 and 5 in Fig. 8(d). When the stored fluid sample is finally needed, it is moved to the target device again by a newly constructed transportation path, shown as path 6 in Fig. 8(e).

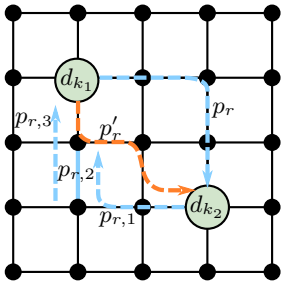

Fig. 9: Connection grid.

Unlike the dedicated storage unit shown in Fig. 1(c), the distributed storage in a channel segment has a higher access efficiency. When a fluid sample stays in a channel segment, that segment is turned into a *storage segment*. When the fluid sample moves again, the segment becomes a part of the transportation path. This concept of channel role switching unifies transportation and storage, and the low-efficiency channels forming storage cells in a dedicated storage unit are excluded completely. In addition, this on-the-spot caching is closer to the target device than a dedicated storage unit, so that the execution efficiency of the assay can also be improved.

To synthesize the architecture of a biochip from its schedule requires to determine the relative locations of devices as well as channel segments and the switches connecting them as shown by the examples in Fig. 8(b)–(e). The devices, switches and their connections together form a *connection graph*. A valid connection graph should be capable of constructing all transportation paths specified in the schedule and caching intermediate fluid samples in channel segments. To reduce resource usage, the synthesized connection graph should contain as few edges as possible.

The connection graph is generated using a general *connection grid*, as shown in Fig. 9. At each node n_i on the grid, either a device or a switch can be assigned. An edge e_j represents a channel segment capable of caching a fluid sample. We use a 0-1 variable $a_{i,k}$ to represent whether device d_k is assigned to node n_i. Since a node can be occupied by no more than one device and a device must be assigned once, $a_{i,k}$ can be constrained as

$$\sum_{d_k \in D} a_{i,k} \leq 1, \forall n_i \in N, \quad \sum_{n_i \in N} a_{i,k} = 1, \forall d_k \in D \qquad (8)$$

where N is the set of nodes in the connection grid and D is the set of devices.

Assume there is a transportation path p_r between device d_{k_1} and device d_{k_2} in the period between t_r^s and t_r^e, where r is the index of the path. t_r^s and t_r^e are constants determined in the scheduling and binding step in Section III-A. We use

a 0-1 variable $\epsilon_{j,r}$ to represent whether the edge e_j is on the path p_r. To construct a path between d_{k_1} and d_{k_2}, we need to guarantee that the path starts from the node for d_{k_1} and ends at the node for d_{k_2}. Consequently, at one of these two nodes, only one of the four edges incident to the node can be covered in the path. At each other node on the path, exactly two edges are covered by the path, as illustrated with path p_r in Fig. 9. Accordingly, we can construct the path with the following constraints

$$\sum_{e_j \in E_i} \epsilon_{j,r} \geq 2 - a_{i,k_1} - a_{i,k_2} - (1 - y_{i,r})M, \quad \sum_{e_j \in E_i} \epsilon_{j,r} \leq y_{i,r} M \tag{9}$$

where E_i is the set of edges incident to node n_i; $y_{i,r}$ is an auxiliary 0-1 variable to indicate whether there is an edge on p_r that is incident to n_i; M is a very large constant to transform the two situations indicated by $y_{i,r}$ into linear constraints [25]. The path construction constraints become more complex when a storage is involved, which leads to three sub-paths: 1) from d_{k_1} to a storage segment; 2) the storage segment caching the fluid sample; 3) from the storage segment to the target device d_{k_2}. We denote the three transportation paths as $p_{r,1}$, $p_{r,2}$ and $p_{r,3}$, as illustrated in Fig. 9. Since the end node of $p_{r,1}$ and the starting node of $p_{r,3}$ can be any node on the connection grid as long as they are the two end nodes of the same edge, we use 0-1 variable a_{i_1,r_1} to represent that node n_{i_1} is the last node on the path $p_{r,1}$ and the variable a_{i_2,r_2} to represent that node n_{i_2} is the first node on the path $p_{r,2}$. Afterwards, we create constraints similar to (9) for each sub path. In addition, we include the constraint that n_{i_1} and n_{i_2} are the two end nodes of the same edge.

In the schedule, there are many transportation paths at a given moment. These paths should not intersect at a node or share the same edge to avoid conflict. Therefore, we examine the paths on the connection grid at the starting time of each transportation path, because this is the moment a new transportation is initiated. At each such moment t, we constrain that all the paths existing on the connection grid should not share any edge or intersect at a node, as,

$$\sum_{e_j \in E} \epsilon_{j,r} \leq 1, \forall p_r \in P_t, \quad \sum_{n_i \in N} \eta_{i,r} \leq 1, \forall p_r \in P_t \tag{10}$$

where P_t is the set of paths existing at time t; $\eta_{i,r}$ represents whether node n_i is on path p_r; E and N are the sets of edges and nodes in the connection grid, respectively. Exception of constraint (10) is that the two ending nodes of the storage segment $p_{r,2}$ can be passed by other paths when the fluid sample is stored, as path p'_r in Fig. 9, so that their variables $\eta_{i,r}$ are not included in (10) when $p_{r,2}$ exists.

When generating the architecture of the chip, we minimize the number of edges that are really used by the transportation paths to reduce resource usage. If an edge is used once by any transportation path, it should appear in the chip. We use a 0-1 variable s_j to represent whether a channel segment should be kept in the chip, and constrain it as

$$s_j \geq \epsilon_{i,r}, \ \forall p_r \in P \tag{11}$$

where P is the set including all transportation paths.

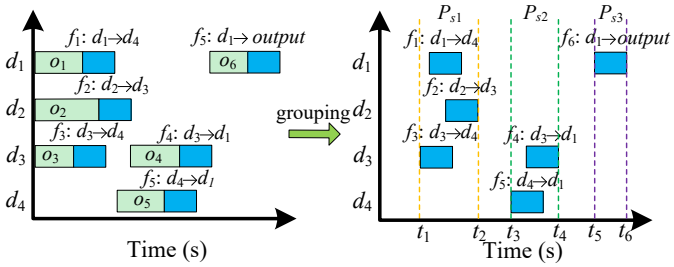

Fig. 10: Illustration of transportation-task grouping during flow-path planning.

Finally, the architecture of the biochip can be determined by solving the following optimization problem

$$\text{minimize} \quad \sum_{e_j \in E} s_j \tag{12}$$

$$\text{subject to} \quad (8)\text{–}(11) \tag{13}$$

After determining s_j, the edges and nodes that are not used in the connection grid are removed to generate the chip architecture as a planar connection graph similar to Fig. 8(b)–(e).

C. Scheduling Recomputation

As discussed previously, since the layout of the biochip has not been determined during the high-level synthesis stage, we use a constant u_c to estimate the durations of transportation tasks. After completing the architectural synthesis above, a biochip architecture that defines the positions of devices and the corresponding connections among them is generated. With this chip architecture, the execution time of each transportation task in the scheduling can be computed directly. Thereafter, the generated chip architecture is fed back into the high-level synthesis stage by re-performing the proposed scheduling method, thus updating the execution procedure of the complete bioassay with accurate transportation latencies. Note that the velocity of fluid flow is set to 10 mm/s in the our method [26].

IV. Flow-Path Planning

In this subsection, the proposed flow-path planning for moving fluids will be discussed in detail. Our goal is to implement all the fluid transportation/caching tasks without any conflict, so that the bioassay can be completed within a short time. Moreover, extra resources such as ports and channel segments introduced to the biochip can also be minimized during the flow-path planning.

A. Flow-Path Planning on A Certian Biochip Architecture

As discussed previously, transportation tasks may be performed concurrently in a scheduling scheme. To avoid conflict among them, the flow paths used for these tasks should avoid to share any on-chip resource, and this therefore requires that all the parallel tasks should be identified from the scheduling before implementing the real flow-path planning.

Accordingly, in the proposed method, we adopt an efficient sweepline algorithm [27] to divide all the transportation tasks in a scheduling into multiple groups, while ensuring that tasks in the same group will be performed in parallel. During the sweeping, a vertical line is used to scan the entire scheduling diagram from left to right along the time axis. Moreover, we maintain an active task group P_s to record all the parallel transportation tasks at the currently scanned time interval. Each time when the sweeping line encounters a transportation task f_r, if $P_s = \emptyset$ or f_r overlaps with at least one task in P_s in their execution time, f_r will be added to P_s. Otherwise, P_s will be stored temporarily and the scanning will be continued using a new active task group with f_r as its first element. As the result of the whole scanning process, several task groups can be generated and each of them consists of a set of parallel transportation tasks. For example, Fig. 10 shows the scheduling scheme of a bioassay, where a total of six transportation tasks (f_1–f_6) are scheduled to complete the execution of specified operations. After implementing the sweepline algorithm discussed above, the transportation tasks in Fig. 10 are divided into the following three groups:

- P_{s1}: {task$|f_1, f_2, f_3$}, (time interval, $t_1 \rightarrow t_2$)
- P_{s2}: {task$|f_4, f_5$}, (time interval, $t_3 \rightarrow t_4$)
- P_{s3}: {task$|f_6$}, (time interval, $t_5 \rightarrow t_6$)

Since transportation conflicts can only occur among parallel tasks, this enables us to deal with the flow-path planning problem according to the task grouping result above. In other words, a flow-path planning solution without transportation conflicts can be obtained by sequentially constructing flow paths for the tasks in each group. Accordingly, based on the connection grid modelled in Fig. 9 as well as the chip architecture generated in Section III-B, we formulate the flow-path planning problem into an ILP model to efficiently find an optimized solution.

Given a task group P_s obtained by the sweepline algorithm, our goal is to find an optimized flow path for each transportation task in P_s, while ensuring that these paths do not share any on-chip resource. More specifically, for a task $f_r \in P_s$ with d_i and d_j as its source and target devices, assume that d_i and d_j are assigned to nodes n_i and n_j in the connection grid. Since the flow path of f_r is required to start with a pump and ends with a waste port while it passes through both d_i and d_j, we use a 0-1 variable $u^r_{n_k}$ to represent whether a node n_k is on the flow path of f_r. Thus we derive the following constraints

$$\sum_{n_k \in N_p} u^r_{n_k} = 1, \ \sum_{n_k \in N_w} u^r_{n_k} = 1 \tag{14}$$

$$u^r_{n_i} = 1, \ u^r_{n_j} = 1 \tag{15}$$

where N_p and N_w are nodes occupied by pump ports and waste ports, respectively. Note that the positions of these ports are scattered to some empty nodes in the connection grid.

On the other hand, if task f_r transports the resulting fluid of device d_i to a channel segment bound to edge e_j for temporary caching, constraint (15) can be reformulated as

$$u^r_{n_i} = 1, \ u^r_{e_j} = 1 \tag{16}$$

where $u^r_{e_j}$ is a 0-1 variable representing whether an edge e_j is on the flow path of f_r.

In addition, channel segments on the chip, in other words edges on the grid, used for connecting the devices and ports on the flow path of f_r need to be determined. Note that at the pump port or the waste port, only one of the four adjacent edges can be covered in the path. At each other device on the path, exactly two adjacent edges are covered by the path. Accordingly, the flow path f_r can be computed with the following constraints

$$\sum_{e_j \in E_{n_p}} u^r_{e_j} - (1 - u^r_{n_p})M \leq 1, \ \forall n_p \in N_p \tag{17}$$

$$\sum_{e_j \in E_{n_p}} u^r_{e_j} + (1 - u^r_{n_p})M \geq 1, \ \forall n_p \in N_p \tag{18}$$

$$\sum_{e_j \in E_{n_w}} u^r_{e_j} - (1 - u^r_{n_w})M \leq 1, \ \forall n_w \in N_w \tag{19}$$

$$\sum_{e_j \in E_{n_w}} u^r_{e_j} + (1 - u^r_{n_w})M \geq 1, \ \forall n_w \in N_w \tag{20}$$

$$\sum_{e_j \in E_{n_k}} u^r_{e_j} - (1 - u^r_{n_k})M \leq 2, \ \forall n_k \in N_d \backslash \{N_p, N_w\} \tag{21}$$

$$\sum_{e_j \in E_{n_k}} u^r_{e_j} + (1 - u^r_{n_k})M \geq 2, \ \forall n_k \in N_d \backslash \{N_p, N_w\} \tag{22}$$

where E_{n_p}, E_{n_w}, and E_{n_k} are the sets of edges adjacent to the nodes n_p, n_w, and n_k, respectively. N_d represents all the nodes occupied by devices in the connection grid. M is a very large constant to transform the formulas above into linear constraints.

Another issue that needs to be considered is that the flow path of f_r should bypass the devices used for other on-going operations as well as the channel segments used for fluid caching. These constraints can be formulated as

$$u^r_{n_k} = 0, \ \forall n_k \in N_b \tag{23}$$

$$u^r_{e_j} = 0, \ \forall e_j \in E_b \tag{24}$$

where N_b is the set of nodes occupied by other active devices and E_b is the set of edges occupied by channels for caching.

Finally, to avoid transportation conflict between flow paths, for any transportation task $f_s \in P_s$, the flow paths of f_s should not share any node on the chip with that of f_r, leading to the following constraint

$$u^r_{n_k} + u^s_{n_k} \leq 1, \ \forall n_k \in N \tag{25}$$

where N is the set of nodes in the connection grid.

In practice, the transportation time of fluids as well as the pressure used for driving the movement of fluids are strongly affected by to the length of flow paths [28]. Hence, this length should be minimized and the exact flow paths for all the transportation tasks can be computed by solving the following optimization problem

$$\text{minimize} \sum_{f_r \in F_t} \sum_{e_j \in E_c} u^r_{e_j} \tag{26}$$

$$\text{subject to (14)–(25)} \tag{27}$$

where F_t is the set of transportation tasks in the scheduling scheme and E_c is the set of edges occupied by channel segments in the connection graph.

B. Deadlock Removal

As discussed in Section II-B, deadlock may occur in a distributed channel-storage system due to the dual functions of flow channels, and this may make the optimization problem defined in (26)–(27) unsolvable. To address the deadlock problem without introducing extra resources to the biochip architecture, fluids cached in those blocked paths need to be moved to other unused channel segments. To this end, we use a 0-1 variable m_j to indicate whether the fluid cached in edge e_j needs to be moved away such that the current deadlock can be eliminated. Accordingly, constraint (24) can be reformulated as

$$u^r_{e_j} - m_i \leq 0,\ \forall e_j \in E_b. \tag{28}$$

Moreover, since moving fluids from a channel segment to another can lead to extra cost such as delay of bioassay, the number of movement operations needs to be minimized. Accordingly, the optimization problem defined in (26)–(27) can be reformulated as

$$\text{minimize} \quad \sum_{f_r \in F_t} \sum_{e_j \in E_c} u^r_{e_j} + \sum_{e_j \in E_b} P_f * m_j \tag{29}$$

$$\text{subject to (14)–(23), (25), (28)} \tag{30}$$

where the penalty factor P_f is a large constant to punish the movement of cached fluids.

After solving the ILP formulation above, fluids that need to be moved away are known. In other words, if a 0-1 variable m_j is set to 1 in the ILP solution, the corresponding fluid cached in channel segment e_j should be removed so that the deadlock can be eliminated. Moreover, since the flow path for each transportation task has already been determined after solving the ILP formulation, new location for caching the removed fluid can be determined by selecting an unused channel segment without introducing new deadlock. Note that the flow path for transporting the removed fluid can also be constructed using the ILP constraints defined in (14)–(25).

C. Conflict Elimination – Scheduling Adjustment

On the other hand, transportation conflict is another factor that may render the path-planning formulation in (26)–(27) unsolvable. Accordingly, in Section II-B, two conflict elimination strategies, i.e., scheduling adjustment and architecture adjustment, are proposed to solve this problem.

As illustrated in Fig. 5, the core idea of scheduling adjustment is to postpone the execution of some transportation tasks, so that those conflicting tasks do not overlap in their execution time. Since postponing a transportation task might trigger a chain reaction such that the execution of the complete bioassay is prolonged, choosing an appropriate postponing scheme with minimized overall time extension is crucial to the whole flow-path planning. For example, Fig. 11 shows two postponing schemes for solving the transportation conflict in Fig. 4. In

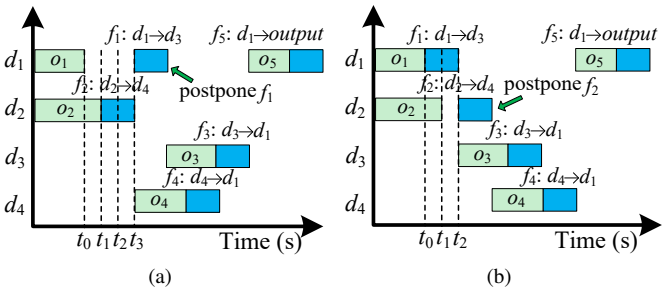

Fig. 11: Comparison on different scheduling-adjustment schemes. (a) Postpone the execution of task f_1. (b) Postpone the execution of task f_2.

Fig. 11(a), the starting time of task f_1 is postponed from t_0 to t_3. Since the result of operation o_1 is directly related to the execution of both o_3 and o_5, this leads to a $t_3 - t_0$ time extension for the complete bioassay. However, in Fig. 11(b), the starting time of f_2 is postponed from t_1 to t_2 and the time extension introduced to the bioassay is only $t_2 - t_1$. Although both schemes can eliminate the current transportation conflict, the goal of scheduling adjustment is to find a solution with minimized extra time extension such as in Fig. 11(b) .

To realize the scheduling adjustment, we formulate the postponing schemes of transportation tasks as well as the corresponding scheduling results into an ILP model. Assume that C is the set of all feasible postponing schemes, we use a 0-1 variable c_j to represent whether the j-th scheme is select to eliminate a transportation conflict. Accordingly, constraint (25) can be reformulated as

$$u^r_{n_k} + u^s_{n_k} + c_j \leq 2,\ \forall n_k \in N,\ \forall c_j \in C \tag{31}$$

$$\sum_{c_j \in C} c_j = 1 \tag{32}$$

where p_r and p_s are two transforation tasks that overlap in their execution time in scheme c_j.

Moreover, to minimize the execution time of the complete bioassay, we adopt the well-known list scheduling algorithm [29] to compute the extra extension of execution time introduced by a postponing scheme, and incorporate the corresponding result into our objective function. Accordingly, the optimization problem defined in (29)–(30) can be further reformulated as

$$\text{minimize} \quad \sum_{f_r \in F_t} \sum_{e_j \in E_c} u^r_{e_j} + \sum_{e_j \in E_b} P_f * m_j + \sum_{c_j \in C} d_j * c_j \tag{33}$$

$$\text{subject to} \quad \text{(14)–(23), (28), (31)–(32)} \tag{34}$$

where d_j is the extra extension of execution time introduced by the postponing scheme c_j.

D. Conflict Elimination – Architecture Adjustment

When biochips are used for large-scale biochemical analysis, a large number of operations need to be executed with limited resources in parallel. Particularly in a distributed channel-storage architecture, a large amount of intermediate

fluids needs to be cached in flow channels temporarily and this further aggravates the conflicts among transportation tasks, making the ILP model formulated in (33)–(34) very difficult to solve. Moreover, the completion of bioassays may also be prolonged severely. Accordingly, in this subsection, we implement the architecture adjustment method proposed in Section II-B by introducing more resources, such as channel segments, pumps, and waste ports, to the chip architecture, so that the large-scale transportation conflicts discussed above can be solved effectively.

To generate a new biochip architecture with minimized extra resources, we apply the heuristic optimization method Genetic Algorithm, which is first introduced by John Holland in 1975 [30]. It is developed based on the concept of Charles Darwin's theory of natural evolution, and has been found to be robust in solving complex engineering and scientific optimization problems.

In the proposed method, a population of candidate solutions, also referred to as chromosomes, is randomly initialized in the whole solution space. The encoding of each chromosome is represented in binary as strings of 0s and 1s. All the chromosomes are then evolved by iteratively updating their encodings using the genetic operators such as mutation and crossover, with the population in each iteration called a generation. In each generation, a fitness value of each chromosome is computed using an evaluation function to assess its solution quality. Finally, the algorithm terminates when a satisfactory fitness level is reached for the population.

Since each chromosome represents an augmented biochip architecture with new introduced resources, its corresponding encoding should include the following information: 1) the locations of newly introduced resources like pumps and waste ports and 2) the locations of newly introduced channel segments. Accordingly, we employ a two-dimensional 0-1 vector $\vec{X} = [\vec{x}^d, \vec{x}^c]$ to encode a chromosome, where vector $\vec{x}^d = [x_0^d, x_1^d, ..., x_m^d]$ is used to denote the locations of new resources in the connection grid, which can be defined as

$$x_i^d = \begin{cases} 1 & \text{if node } n_i \text{ is occupied by a new resource} \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

Vector $\vec{x}^c = [x_0^c, x_1^c, ..., x_n^c]$ is used to indicate the locations of new channel segments in the connection grid, which can be defined as

$$x_j^c = \begin{cases} 1 & \text{if edge } e_j \text{ is assigned to a new channel segment} \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

In addition, the fitness of a chromosome is evaluated by three criteria during the evolution of population: 1) if the biochip architecture represented by a chromosome is invalid, in other words, if we still cannot find a feasible flow-path planning solution on the new chip, the fitness of this chromosome is set to $-\infty$, meaning that the newly introduced resources still cannot eliminate all the transportation conflicts, 2) if flow paths for all the transportation tasks can be constructed on the new chip architecture without any conflict, the algorithm tends to select the chromosome that leads to shorter execution time of the bioassay, and 3) our method tends to select the chromosome with minimized extra resources. Accordingly, the evaluation function used in the proposed algorithm can be formulated as

$$F(X_i) = \begin{cases} -\infty & \text{if } X_i \text{ is invalid} \\ C_1 - C_2 \times N_d - C_3 \times T_e & \text{otherwise} \end{cases} \tag{37}$$

where C_1, C_2, and C_3 are three constants, N_d represents the number of extra resources introduced to the chip, and T_e represents the execution time of the bioassay.

The detailed steps of the proposed architecture adjustment method can be summarized as follows:

1) Based on the previously generated biochip architecture in the connection grid, chromosomes are initialized by randomly introducing new resources as well as channel segments to the chip. These chromosomes form the first generation of the population.
2) For each chromosome X_i in the population, the proposed flow-path planning method is employed to evaluate the biochip architecture represented by X_i and generate a fitness value $F(X_i)$ according to (37).
3) All the chromosomes are sorted in non-increasing order according to their fitness values, and the first half of the population is then selected as the parent of next generation.
4) Genetic operators including mutation and crossover are applied to the selected chromosomes to generate new chip architectures. Mutation operation generates a new chromosome by reversing the values of some randomly selected encoding positions on a parent chromosome. Crossover operation exchanges information of some encoding positions between two parent chromosomes. Both the parent and child chromosomes form the next generation of population.
5) Step (2) to (4) are executed repeatedly until the maximum allowed number of iterations is reached. The biochip architecture represented by the chromosome with the best fitness value is then selected as the final solution.

TABLE II: Details of the benchmarks used in this paper.

Benchmark	$\|O\|$	n_i	n_o	D_p	D_e
RA100	100	6	6	16	5 mixers
RA70	70	7	8	10	3 mixers, 2 detectors
CPA	55	7	8	9	3 mixers, 2 detectors
RA30	30	11	1	7	3 mixers, 1 detector, 1 filter
IVD	12	6	6	3	3 mixers, 2 detectors
PCR	7	4	1	3	2 mixers

V. Simulation Results

The proposed method was implemented in C++ and tested on a PC with 3.20 GHz CPU and 8 GB memory. We demonstrate the results using three real-life assays and three randomly generated assays. The information of these test cases is shown in Table II, where CPA (Colorimetric Protein Assay), IVD (In-Vitro Diagnostics) and PCR (Polymerase Chain Reaction) are real-world assays [31] and the other three assays are randomly generated. The Column $|O|$ in Table II

TABLE III: Results of Scheduling, Architectural Synthesis, and Flow-Path Planning

Benchmark	Scheduling			Architectural Synthesis					Flow-Path Planning							
	c_t	t^E (s)	t_s (s)	G	n_e	n_v	t_r (s)	t^{Eu}(s)	n_p	n_w	n_{tv}	n_{cr}	n_{tj}	l_n	$t^{E'}$ (s)	t_p (s)
RA100	50	182	1800	5×5	32	58	1867	230	2	2	107	16	9	420	272	10872
RA70	32	118	1800	4×4	20	38	1819	156	2	2	82	9	10	270	183	2562
CPA	33	107	1800	4×4	20	40	1817	121	2	2	72	8	8	220	152	2455
RA30	3	67	300	4×4	8	16	1800	75	2	2	43	3	5	170	81	821
IVD	0	28	< 1	4×4	5	10	25	44	2	2	32	1	4	110	47	580
PCR	2	29	< 1	4×4	5	8	20	32	2	2	18	0	2	90	32	133

c_t: the total amount of time that fluids are stored. t^E: execution time of the assay after scheduling.
t_s: runtime of scheduling. G: size of the connection grid. n_e: the number of channel segments. n_v: the number of valves.
t_r: runtime of architectural synthesis. t^{Eu}: execution time of the assay after scheduling recomputation.
n_p: the number of pumps. n_w: the number of waste ports. n_{tv}: the total number of valves after performing the architectural adjustment. n_{cr}: the number of channel crossings. n_{tj}: the number of T-junctions. l_n: the total length of the flow channels.
$t^{E'}$: final execution time of the assay. t_p: runtime of flow-path planning.

shows the number of operations in each assay. The Columns n_i and n_o are the number of inputs and outputs of each assay, respectively. The Column D_p gives the depth of the sequencing graph corresponding to each assay. The Column D_e lists the number of available devices for each assay, respectively. The execution latencies of mixing, detection, and filtration operations are set to 4 s, 7 s, and 3 s in the benchmarks, respectively [31]. Moreover, in the experiments, we used Gurobi [32] to solve the optimization problems.

A. *Synthesis Results of the Proposed Design Flow*

In this part, we first report the synthesis results of each stage in the proposed design flow in Table III, including scheduling, architectural synthesis, as well as flow-path planning.

The result of scheduling with storage minimization is shown in the columns c_t, t^E and t_s, where c_t is the total amount of time that fluids are stored, t^E is the execution time of the assay defined in (5), and t_s is the runtime for solving the optimization problem (6)–(7), which was limited to 30 minutes for the solver to return the best-effort results. It can be seen that the execution of each bioassay is completed within a reasonable time.

In architectural synthesis, we use a connection grid to determine device locations and channel segments connecting them. The size of the grid is shown in column G. Moreover, the numbers of channel segments and valves in the chip are shown in the columns n_e and n_v, respectively. Note the valves counted in the experiments do not include those used in mixers. The runtimes to generate chip architectures are shown in the column t_r. Moreover, with the generated chip architecture, we re-perform the proposed scheduling method to map the bioassay onto the synthesized architecture, thus generating an updated scheduling with accurate transportation latencies. Correspondingly, the updated execution times of assays are shown in the column t^{Eu}.

After flow-path planning, the total number of valves in the chip is reported in the column n_{tv}. It can be seen that new valves are introduced into the chip. We analyze this for two reasons: 1) since pumps and waste ports are added to the chip in this stage, these devices should be connected to the previously allocated devices such as mixers by constructing new flow channels. These channels may intersect with previously constructed flow paths, thereby leading to some extra valves and 2) the number of pumps and waste ports introduced to the chip is minimized to reduce extra chip cost (see columns n_p and n_w), thus increasing the number of new constructed channels. The numbers of crossings and T-junctions in the final chip architecture are shown in the columns n_{cr} and n_{tj}, respectively.

Moreover, the total length of flow channels in the final chip is reported in the column l_n. The final execution time of the assay is shown in the column $t^{E'}$, which is a little bit longer than the one shown in the column t^{Eu} due to the effects of the proposed scheduling adjustment method. Finally, the runtimes of the flow-path planning are shown in the column t_p, which are acceptable for a high-quality transportation network without any conflict.

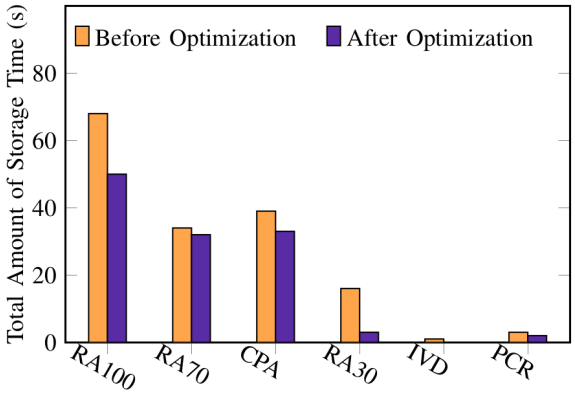

Fig. 12: Comparison of the total amount of time that fluids are stored before and after applying the proposed storage optimization method.

B. *Performance Validation of the Proposed Optimization Methods*

In this subsection, the performance of several optimization methods in the proposed synthesis flow is validated in detail.

We first evaluate the effectiveness of the proposed storage optimization method in scheduling. Fig. 12 shows the comparison results on the total amount of time that fluids are stored

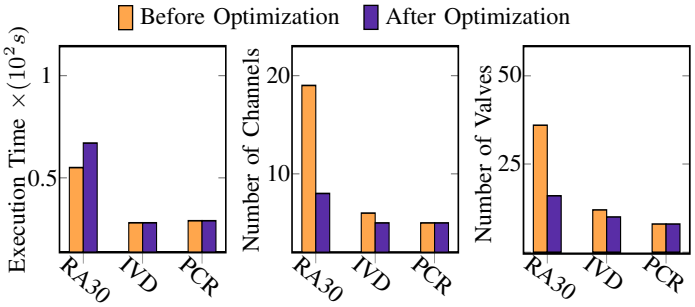

Fig. 13: Comparison results in terms of the execution time of assay and the numbers of routing channels and valves before and after applying the proposed storage optimization method.

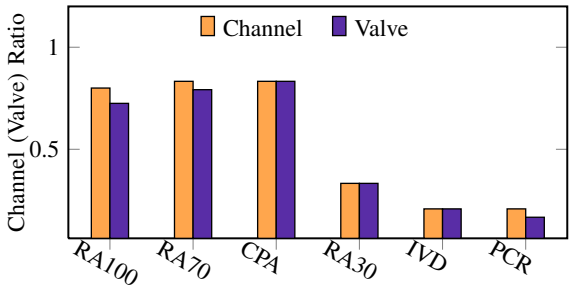

Fig. 14: Channel and valve ratios with respect to the results of architectural synthesis and the original channels and valves in the connection grid.

before and after incorporating this method into the scheduling. It can be seen that the total amount of time that fluids are stored in each assay is reduced significantly after applying the proposed optimization method. Moreover, Fig. 13 shows the comparison of execution time of assays, the number of routing channels, and the number of valves in the two cases with and without considering storage optimization. It can be observed that storage optimization generated comparable execution time in the cases IVD and PCR, but the execution time of RA30 is slightly increased, which is acceptable for most biochemical experiments. However, the numbers of routing channels and valves in the result of RA30 are much smaller, because storage optimization is partially achieved by reducing the number of fluid-storage operations, thus reducing the numbers of routing channels and valves that need to be constructed during architectural synthesis.

In architectural synthesis, we start with a connection grid. After synthesis, only the channels that are used at least once are kept in the result. The ratios of the number of used channels to the total number of all the channels in the grid is shown in Fig. 14, where all these ratios are smaller than 1, and a half of them are even below 0.5, indicating that the architectural synthesis approach confines resource usage effectively on only a part of channels. After removing the unused channels, the number of valves is also reduced, as shown by the valve ratios in Fig. 14.

To validate the efficiency of the proposed distributed channel-storage architecture, the results of comparing execution time and number of valves for our method and the method using a dedicated storage are reported in Fig. 15. The ratios are computed as our results/the results using dedicated storage.

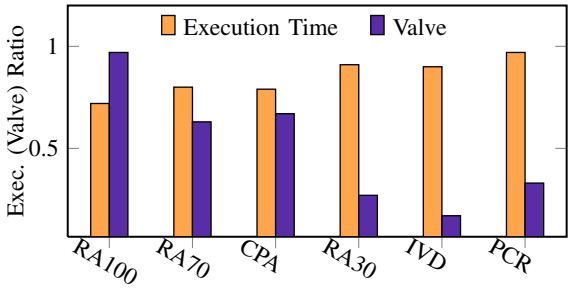

Fig. 15: Execution time and valve ratios with respect to the results using distributed channel storage and dedicated storage units.

TABLE IV: Comparison with the method in [26].

Benchmark	Minhass [26]			Proposed method		
	$t^{E'}$ (s)	n_v	l_n	$t^{E'}$ (s)	n_v	l_n
RA70	238	117	286	183	82	270
CPA	185	93	227	152	72	220
RA30	99	59	188	81	43	170
IVD	42	37	137	47	32	110
PCR	33	18	106	32	18	90

It can be seen from Fig. 15 that channel storage architecture leads to a relatively higher execution efficiency of bioassays, mainly due to the following reasons: 1) channel storage can be considered as distributing the storage cells of a dedicated storage to the whole chip plane in an optimized manner, leading to an "on-the-spot" fluid store/fetch with high efficiency, 2) in the proposed method, by constructing storage channels at appropriate positions while finding optimized flow paths for fluid transportation tasks, congestion problem can be alleviated effectively, 3) when using a dedicated storage with fixed position, conflicts can also occur between store/fetch operations and fluid transportation tasks since they may share the same flow channels during their execution, and 4) the structure of a dedicated storage (see Fig. 1(c) and Fig. 3(c)), as discussed in Section I, suffers from the drawback of bandwidth limitation.

Additionally, from the comparison in Fig. 15, we can see that the number of valves used in the chip is also reduced in the channel storage architecture. This is mainly because: 1) a dedicated storage usually contains quite a number of valves inside the multiplexer-like ports as shown in Fig. 1(c) and 2) although two valves should be placed at the ends of a storage channel, these valves can be saved in most cases by sharing the valves placed at channel crossings.

Moreover, in [26] a top-down synthesis flow is proposed to generate chip architectures with minimized application completion time. This method, however, still adopts the traditional dedicated storage to deal with the intermediate fluids generated during the execution of bioassays. To further verify the effectiveness of the proposed method and the distributed channel-storage architecture, we implemented the method in [26] and

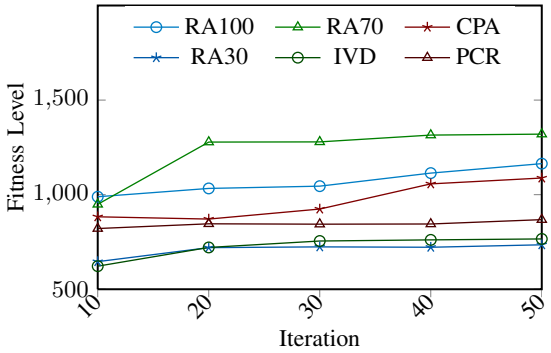

Fig. 16: Average convergence curves of the proposed genetic algorithm.

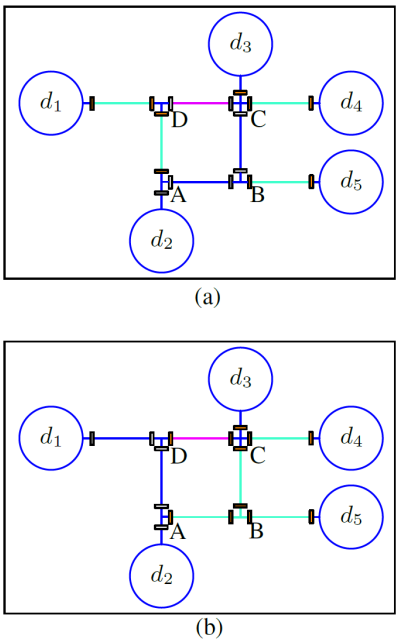

Fig. 17: Snapshots of the synthesized chip executing RA30 at different times. The channel segments in blue are transporting and in purple are caching fluid samples.

ran both algorithms on the same benchmarks. Table IV shows the comparison results, where the columns $t^{E'}$, n_v and l_n represent the completion time of the bioassay, the number of valves in the chip, and the total channel length.

It can be seen from Table IV that the proposed algorithm achieves shorter completion time of the bioassay on most test cases. There are two major reasons for these resutls: 1) dedicated storage suffers from the drawbacks of bandwidth limitation and fixed position on the chip as discussed in Section I, leading to a low efficiency of fluid store and fetch. In contrast, by caching fluids on-the-spot with distributed channel storage and coordinating the relations between fluid caching and transportation without any conflict, the efficiency of fluid store and fetch can be improved significantly and 2) in [26] the traditional heuristic list-scheduling method is adopted to compute a feasible scheduling scheme. The solution quality, however, is limited due to the greedy strategy adopted in the list scheduling. On the other hand, the proposed method further

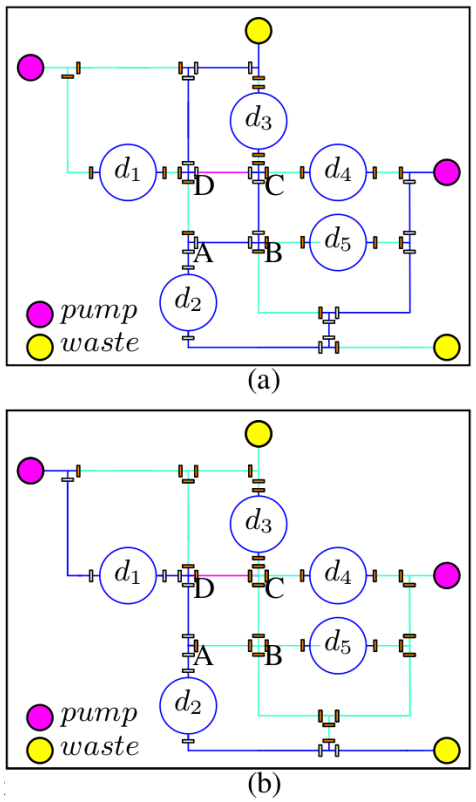

Fig. 18: Snapshots of the synthesized chip executing RA30 with constructed flow paths at different times. The channel segments in blue are transporting and in purple are caching fluid samples.

takes the storage optimization into account in the high-level synthesis stage to reduce the total amount of storage time, thus improving the execution efficiency of the assay. Moreover, the number of valves used in the proposed method is fewer than that of [26] due to the following reasons: 1) independent flow paths need to be constructed between devices and dedicated storages, leading to more channel crossings and 2) a dedicated storage also requires quite a number of valves to direct the fluid flow as can be seen from Fig. 1(c). Also, the proposed method leads to an overall shorter channel length as can be seen from Table IV.

In the flow-path planning stage, since the proposed genetic algorithm plays an important role in determining the new biochip architecture, to evaluate its efficiency, Fig. 16 illustrates the corresponding convergence curves with respect to different bioassays. It can be seen that the average fitness level of the population is improved as iteration time increases, and the proposed algorithm reaches a convergence quickly in most of the test cases.

C. *Snapshots of Assay Execution*

Finally, We show the execution snapshots of the assay RA30 in Fig. 17 and Fig. 18, respectively. Fig. 17 demonstrates the fluid transportation without considering the actual flow paths construction. In Fig. 17(a), a transportation path is formed as $d_2 \rightarrow A \rightarrow B \rightarrow C \rightarrow D$ to store a fluid sample

into the channel segment between C and D. In Fig. 17(b), a transportation path is constructed as $d_1 \rightarrow D \rightarrow A \rightarrow d_2$ while the channel segment between C and D is caching a fluid sample. Fig. 18 demonstrates the fluid transportation on the chip architecture after completing the flow-path palnning. In Fig. 18(a), a flow path is formed as $pump \rightarrow d_2 \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow waste$ to store a fluid sample into the channel segment between C and D. In Fig. 18(b), a flow path is formed as $pump \rightarrow d_1 \rightarrow D \rightarrow A \rightarrow d_2 \rightarrow waste$, while an intermediate liquid is stored between C and D.

VI. Conclusion

We have proposed the first method to generate a biochip architecture considering storage optimization. By caching fluid samples on-the-spot, fluid transportation and storage can be performed in a unified manner, leading to an efficient distributed channel-storage architecture without traditional dedicated storage unit. With this new architecture, we proposed a systematic method that considers the scheduling of biochemical operations, placement of allocated devices, as well as routing of transportation channels simultaneously. Moreover, after completing the architectural synthesis, transportation tasks were mapped to the generated chip dynamically to realize the actual manipulation of fluid transportation/caching without any conflict. This is also the first work to consider the flow-path planning of transportation tasks in a distributed channel-storage biochip architecture. Simulation results confirmed that with this uniform model the architecture generated by the proposed method is more efficient in executing bioassays.

References

[1] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann, "Transport or store?: Synthesizing flow-based microfluidic biochips using distributed channel storage," in *Proc. Design Autom. Conf.*, 2017, pp. 49:1–49:6.

[2] E. Verpoorte and N. F. D. Rooij, "Microfluidics meets MEMS," *Proc. IEEE*, vol. 91, no. 6, pp. 930–953, 2003.

[3] J. Melin and S. Quake, "Microfluidic large-scale integration: the evolution of design rules for biological automation," *Annu. Rev. Biophys. Biomol. Struct.*, vol. 36, pp. 213–231, 2007.

[4] K. Hu, K. Chakrabarty, and T.-Y. Ho, *Computer-Aided Design of Microfluidic Very Large Scale Integration (mVLSI) Biochips*. Springer, 2017.

[5] E. P. Kartalov, J. F. Zhong, A. Scherer, S. R. Quake, C. R. Taylor, and W. French Anderson, "High-throughput multi-antigen microfluidic fluorescence immunoassays," *BioTechniques*, vol. 40, no. 1, pp. 85–90, 2006.

[6] S. Einav, D. Gerber, P. D. Bryson, E. H. Sklan, M. Elazar, S. J. Maerkl, J. S. Glenn, and S. R. Quake, "Discovery of a hepatitis c target and its pharmacological inhibitors by microfluidic affinity analysis," *Nat. Biotechnol.*, vol. 26, no. 9, pp. 1019–1027, 2008.

[7] J. W. Hong, Y. Chen, W. F. Anderson, and S. R. Quake, "Molecular biology on a microfluidic chip," *J. Phys.-Condes. Matter.*, vol. 18, no. 18, pp. 691–701, 2006.

[8] Y. Zhu, B. Li, T.-Y. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann, "Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips," in *Proc. Int. Conf. Comput.-Aided Des.*, 2018, pp. 123:1–123:8.

[9] T. M. Squires and S. R. Quake, "Microfluidics: Fluid physics at the nanoliter scale," *Revi. Modern Physi.*, vol. 77, no. 3, pp. 977–977, 2005.

[10] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer routing and control-pin minimization for flow-based microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 1, pp. 55–68, 2017.

[11] N. Amin, W. Thies, and S. P. Amarasinghe, "Computer-aided design for microfluidic chips based on multilayer soft lithography," in *Proc. Int. Conf. Comput. Des.*, 2009, pp. 2–9.

[12] Z. Chen, X. Huang, W. Guo, B. Li, T.-Y. Ho, and U. Schlichtmann, "Physical synthesis of flow-based microfluidic biochips considering distributed channel storage," in *Proc. Design, Autom., and Test Europe Conf.*, 2019, pp. 1530–1525.

[13] T. Zhang, R. B. Fair, and K. Chakrabarty, *Microelectrofluidic Systems: Modeling and Simulation*. CRC Press, 2002.

[14] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," in *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embed. Sys.*, 2012, pp. 181–190.

[15] C. Lin, C. Liu, I. Chen, D. T. Lee, and T. Ho, "An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips," in *Proc. Design Autom. Conf.*, 2014, pp. 141:1–141:6.

[16] X. Huang, T.-Y. Ho, K. Chakrabarty, and W. Guo, "Timing-driven flow-channel network construction for continuous-flow microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2019, doi: 10.1109/TCAD.2019.2912936.

[17] Q. Wang, Y. Ru, H. Yao, T. Ho, and Y. Cai, "Sequence-pair-based placement and routing for flow-based microfluidic biochips," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2016, pp. 587–592.

[18] W. H. Minhass, P. Pop, J. Madsen, and T. Ho, "Control synthesis for the flow-based microfluidic large-scale integration biochips," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2013, pp. 205–212.

[19] Q. Wang, S. Zuo, H. Yao, T. Ho, B. Li, U. Schlichtmann, and Y. Cai, "Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2017, pp. 524–529.

[20] H. Yao, T. Ho, and Y. Cai, "PACOR: practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips," in *Proc. Design Autom. Conf.*, 2015, pp. 142:1–142:6.

[21] X. Huang, T.-Y. Ho, W. Guo, B. Li, and U. Schlichtmann, "Minicontrol: synthesis of continuous-flow microfluidics with strictly constrained control ports," in *Proc. Design Autom. Conf.*, 2019, pp. 145:1–145:6.

[22] H. Yao, Q. Wang, Y. Ru, Y. Cai, and T. Ho, "Integrated flow-control codesign methodology for flow-based microfluidic biochips," *IEEE Design & Test*, vol. 32, no. 6, pp. 60–68, 2015.

[23] T. Tseng, M. Li, B. Li, T. Ho, and U. Schlichtmann, "Columba: Co-layout synthesis for continuous-flow microfluidic biochips," in *Proc. Design Autom. Conf.*, 2016, pp. 147:1–147:6.

[24] F. Su and K. Chakrabarty, "Architectural-level synthesis of digital microfluidics-based biochips," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 223–228.

[25] D. Chen, R. Batson, and Y. Dang, *Applied Integer Programming: Modeling and Solution*. Wiley, 2011.

[26] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2012, pp. 181–190.

[27] J. Lane and L. Carpenter, "A generalized scan line algorithm for the computer display of parametrically defined surfaces," *Computer Graphics and Image Processing*, vol. 11, no. 3, pp. 290–297, 1979.

[28] K. Hu, B. B. Bhattacharya, and K. Chakrabarty, "Fault diagnosis for leakage and blockage defects in flow-based microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 7, pp. 1179–1191, 2015.

[29] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[30] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[31] P. Pop, W. H. Minhass, and J. Madsen, *Microfluidic Very Large Scale Integration (VLSI)*. Springer, 2016, pp. 60–62.

[32] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013. [Online]. Available: http://www.gurobi.com

[33] Y. Lim, A. Kouzani, and W. Duan, "Lab-on-a-chip: a component view," *Microsystem Technologies*, vol. 16, no. 12, pp. 1995–2015, 2010.