Block-Flushing: A Block-based Washing Algorithm for Programmable Microfluidic Devices

Yu-Huei Lin[1], Tsung-Yi Ho[1,2], Bing Li[3], Ulf Schlichtmann[3]
[1]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[2]Institute for Advanced Study, Technical University of Munich [3]Chair of Electronic Design Automation, Technical University of Munich

***Abstract*—Programmable Microfluidic Devices (PMDs) have emerged as a new architecture for next-generation flow-based biochips. These devices can be dynamically reconfigured to execute different bioassays flexibly and efficiently owing to their two-dimensional regularly-arranged valve structure. During execution of a bioassay or between the execution of multiple bioassays, some areas on the PMD, however, become contaminated and must be cleaned by washing them with a buffer flow before they are reused. In this paper, we propose a novel block-based washing technique called block flushing. In this method, contaminated areas are first collected according to given patterns and flushed as a whole to increase washing efficiency. Simulation results show that with this technique the proposed method can achieve on average 28% improvement in reducing washing time compared with two other baseline solutions.**

I. Introduction

Microfluidic biochips have drawn much attention in recent years due to their high integration and efficiency to improve experiment flows in traditional laboratories [1]. In a flow-based microfluidic biochip, the movement of continuous flows is controlled by valves. The structure of a valve is shown in Fig. 1(a). In the flow layer, a flow channel for transporting fluids is constructed on a substrate. In the control layer, which is above the flow layer, a control channel is constructed and connected to an air pressure source [2]. The control channel can be dilated after being filled with an air pressure to squeeze the flow channel to block the fluid movement. If the pressure in the control channel is released, the fluids in the flow channel can resume their movement.

Valves can also be used to create complex devices such as mixers and storage units etc. The structure of a mixer is shown in Fig. 1(b). If the three valves on the top of the mixer in Fig. 1(b) are actuated alternatively, a circular flow can be generated to mix the samples and reagents contained inside the mixer. Thereafter, the results of mixing can be transported to other devices for further reaction operations or stored temporarily in storage units until they are needed later.

The nine valves inside the mixer in Fig. 1(b) are operation-specific. The three valves at the top switch very fast to mix fluid samples, while the six valves on the left and the right entrances of the mixer are only used to guide fluid samples to enter or leave the mixer, leading to a low actuation efficiency. To balance the efficiency of valves, Programmable Microfluidic Devices (PMDs) have been proposed [4], [5], as shown in Fig. 1(c). In this architecture, valves are arranged regularly in horizontal and vertical directions. By switching on and off a given set of valves, both the transportation and mixing functions can be implemented, as shown in the videos in [6], [7]. This architecture is highly flexible, since any area on the chip can be used to implement any transportation or mixing function. In addition, it provides the ability of fault tolerance, since an area with manufacturing defects can easily be tolerated by moving operations to another area on the chip.

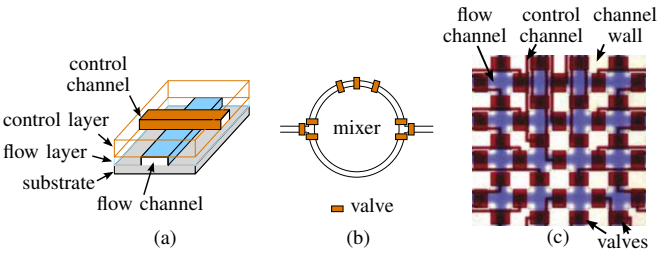


Fig. 1. (a) Valve structure [3]. (b) Mixer. (c) Structure of Programmable Microfluidic Device (PMD) [4].

To take advantage of the reconfigurability of PMDs, several methods have been proposed to execute bioassays on such a regular valve array. In [8], an ILP model is proposed to map operations onto a PMD by reconfiguring different types of devices dynamically while reducing the worst-case wearout. Since PMDs can tolerate manufacturing faults, post-manufacturing test strategies have also been proposed in [3]. In addition, flow routing considering pressure-routes in establishing flow transportation paths on a PMD is discussed in [9] to achieve a better assay completion time. Furthermore, a close-to-optimal physical design solution can also be achieved by adapting the formulation based on satisfiability in [10]. Moreover, reliability of the control logic in such chips is improved in [11].

The methods above mainly focus on functional mapping of bioassays onto PMDs. However, the contamination problem caused by area reuse on such a chip has not been addressed. Though a washing method has been proposed for traditional flow-based biochips [12], it cannot deal with the large number of possible paths in PMDs. On a PMD, some valves can be opened simultaneously to flush connected areas instead of paths. In this paper, we propose an efficient algorithm to merge contaminated areas using a data structure called flushing tree. Thereafter, these merged areas are flushed as a whole to improve washing efficiency.

The rest of this paper is organized as follows. In Section II, we introduce the flushing operation as an advanced washing technique for PMDs and formulate the washing problem. In Section III, we describe our ideas to apply flushing operations

978-3-9819263-2-3/DATE19/©2019 EDAA

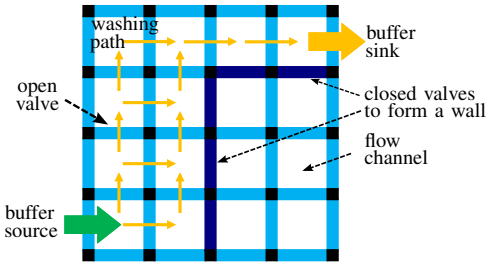


Fig. 2. Flushing operation.

properly by building up flushing trees. In Section IV, we explain how to solve the contamination problem with an algorithm to bind flushing trees. Simulation results are reported in Section V and we conclude this paper in Section VI.

II. Flushing Operation and Problem Formulation

In this section, we first introduce a new washing technique called flushing operation, which can save much time in washing contaminated areas on a PMD compared to ordinary washing operations. Then we introduce the routing problem in washing PMDs. At the end of this section, we define the problem formulation of this paper.

A. Flushing Operation

After operations are executed on a PMD, some areas on the chip become contaminated. A buffer flow can then be applied to remove the contaminants. Flushing, or block-based washing, is a washing strategy collecting used areas on the chip together and washing them as a whole. Unlike flow paths built up for transporting samples and reagents, washing operations on a PMD can be merged in a blockwise manner instead of pathwise. On a PMD, we can close some valves simultaneously to form a wall, as shown in Fig. 2. Afterwards, the left half of the chip can be viewed as a polluted area. Then all valves located in this isolated area can be opened altogether and a buffer fluid can be injected to flush this entire area simultaneously. In this case, the buffer flow injected into the area can be viewed as lots of washing paths spanning from the source to the sink with the shortest distance. This combined washing scheme can significantly increase the washing efficiency.

B. Buffer Fluid Routing on PMDs

A buffer flow cleans all the areas which it traverses. Therefore, we try to generate the buffer flow to cover as many washing targets as possible. To create such a buffer flow, washing paths need to be established from the buffer reservoir, also called buffer source, to the buffer sink.

Since a buffer flow originating from the source becomes contaminated after reaching the first area to be washed, it is not preferable to use it to clean the next area again. Otherwise, a long washing time is needed to finish the washing operation, because the buffer flow becomes clean enough to wash the second area only after the first area is completely cleaned. Accordingly, we need to ensure that a buffer flow containing clean buffer fluid does not intersect with other contaminated buffer flows. Moreover, different from building paths for transporting samples or reagents, washing paths can be connected together

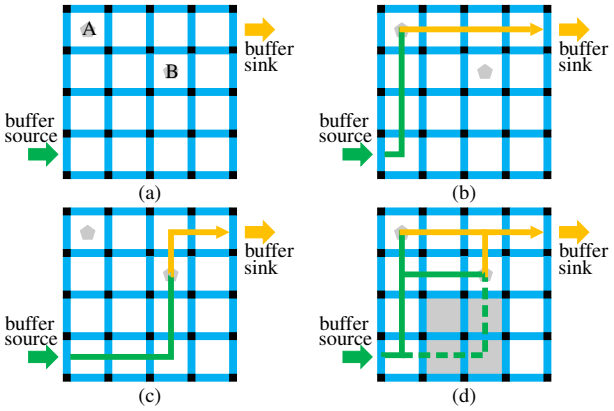


Fig. 3. Merging of buffer flow paths for washing. The buffer flow in green is clean. After a clean buffer flow reaches the first contaminated spot, it becomes contaminated and thus cannot be used to clean the next spot in serial. (a) A chip with two contaminated spots A and B. (b) Buffer flow path to wash spot A. (c) Buffer flow path to wash spot B. (d) Merged buffer flow paths to form a washing tree. The gray area can be spared from the washing process.

if they are all clean or contaminated. With this path merging, the areas that are affected by the washing operations can be reduced.

An example of the merging of washing paths is shown in Fig. 3, where the two buffer flows in Fig. 3 (b)–(c) can wash the spots A and B, respectively. As discussed above, these two spots are not washed in serial in this example to improve washing efficiency. These two buffer flow paths can be combined to form the solution in Fig. 3(d) to flush the spots A and B. Consequently, the gray area in Fig. 3(d) can be spared from the washing process. The solution in Fig. 3(d) also hints that a short spanning tree is efficient in washing contaminated areas. This spanning tree may cover not only the contaminated areas but also clean areas to reduce its height. To construct such a spanning tree, the original buffer flow path to wash spot B in Fig. 3(c) is pushed upwards and merged with the path in Fig. 3(b) to form the solution in Fig. 3(d).

C. Problem Formulation

After executing operations of bioassays, a PMD contains contaminated areas or spots. These contaminated areas and spots can be viewed as washing blocks that need to be washed. In order to improve washing efficiency, these washing blocks should be collected to form larger flushing areas. This problem can be formulated as follows:

- ***Input:*** A PMD architecture; the locations of washing blocks; the locations of the buffer fluid source and sink.
- ***Output:*** The scheme of combination of washing blocks for block-based flushing; the total washing time, which is the sum of fluidic execution time for washing the contaminated areas.
- ***Objective:*** Minimize washing time; reduce the number of flushing operations; avoid clean buffer flows crossing contaminated buffer flows during washing.

III. Basic ideas and flushing tree definition

In this section, we first introduce the concept of washing blocks and their relation to contaminated areas. Afterwards,

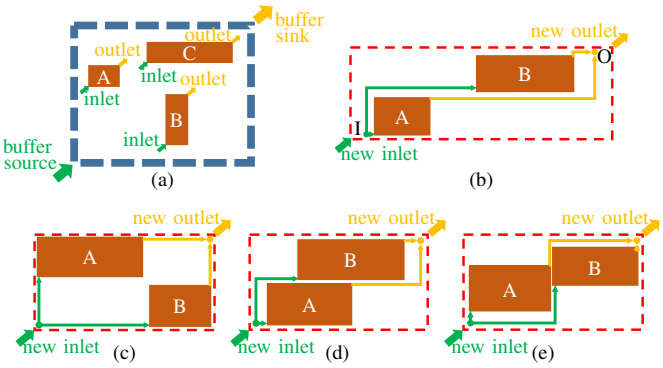


Fig. 4. (a) Washing blocks. (b) The combination of washing blocks A and B. (c) L shape of inlet and outlet. (d) C shape of inlet and outlet. (e) U shape of inlet and outlet.

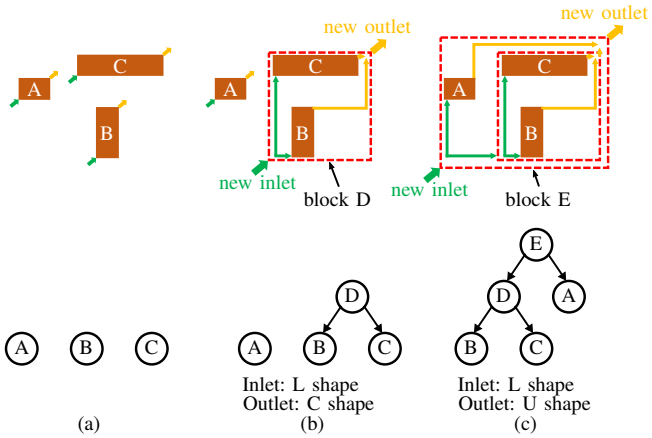


Fig. 5. Construction of flushing trees. (a) Original washing blocks as the initial flushing trees. (b) Bind block B and block C into a new flushing tree D. The new inlet is determined in L shape and the new outlet is determined in C shape. (c) Bind block A and block D into a new flushing tree E. The new inlet is determined in L shape and the new outlet is determined in U shape.

we introduce the concept of pattern combination for collecting washing blocks in given regular patterns. Last, we define a new data structure called flushing tree, following the pattern combination concept, to collect washing blocks.

A. Washing Blocks

To wash contaminated areas, we first transform these areas into washing blocks. A washing block covers a contaminated area as well as neighboring cells to form a rectangular shape. Each washing block has an inlet and an outlet. The inlet is the nearest point to the buffer source on the washing block, and the outlet is the nearest point to the buffer sink on the washing block. A buffer fluid can be injected into a washing block from the inlet and driven out from the outlet. Fig. 4 (a) shows several cases of washing blocks.

B. Pattern Combination

To establish the washing relation to construct washing flows, we combine different washing blocks hierarchically into a bigger block by connecting their inlets and outlets, recursively. Correspondingly, the location to which the inlets of the washing blocks are connected becomes the new inlet for the larger new washing block. Similarly, the new outlet is the connecting point of all the outlets of the original washing blocks. As shown in Fig. 4(b), a larger washing block is constructed with the new inlet set at I and new outlet at O.

During construction of washing blocks, combining patterns can be classified into three categories as follows:

- ***L shape:*** inlets/outlets of blocks A, B, and the new inlet/outlet are connected in L shapes/reversed L shapes, as shown in Fig. 4(c).
- ***C shape:*** inlets/outlets of blocks A, B, and the new inlet/outlet are connected in C shapes/reversed C shapes, as shown in Fig. 4(d).
- ***U shape:*** inlets/outlets of blocks A, B, and the new inlet/outlet are connected in U shapes/reversed U shapes, as shown in Fig. 4(e).

Following the patterns above, the hierarchical construction ensures that no washing blocks are connected in serial to avoid a longer washing time as described previously. In addition, the routing of the flow paths inside a new washing block using the three patterns is confined into a limited area, so that the routing complexity is reduced. This hierarchical construction can be applied recursively until all the washing blocks are connected into a flushing tree.

C. Flushing Tree

To flush several washing blocks as a whole, a new data structure, *flushing tree*, can be applied to collect washing blocks. Flushing tree is a full binary tree, the leaves of which represent washing blocks on a PMD. Each internal node in the flushing tree represents a washing block covering its child trees. With this data structure, we can collect washing blocks by merging different trees into a new flushing tree and storing the result in its root. This flushing tree can also be viewed as the representation of a massive washing block.

An example of merging flushing trees is shown in Fig. 5. In this case, the primary buffer source and buffer sink are located at the lower left corner and the upper right corner of the PMD, respectively. Three contaminated areas, A, B and C, need to be washed, as shown in Fig. 5(a). Since only the three patterns, L, C and U shapes, are applied, the preferable flushing tree is constructed by combining B and C first, as shown in Fig. 5(b). Thereafter, the washing block A is combined with the flushing tree from the previous step to form a solution to flush the three original washing blocks as a whole.

IV. Flushing Tree Construction

The overall flow of constructing the flushing tree from washing blocks using the patterns described above is summarized in Fig. 6. In pair-queue construction, we generate a priority queue from the current flushing trees. This queue contains the pairs of flushing trees as candidates that can be selected in the next stage. Afterwards, we sequentially choose the pairs in this queue to perform pattern routing. If a feasible pair can be found, we merge the two flushing trees corresponding to this pair into a new tree. The original trees that are the children of the new tree are removed so that a washing block appears in the flush tree only once. Congestion resolving is applied if there is no feasible pair. The details of each phase are explained in the

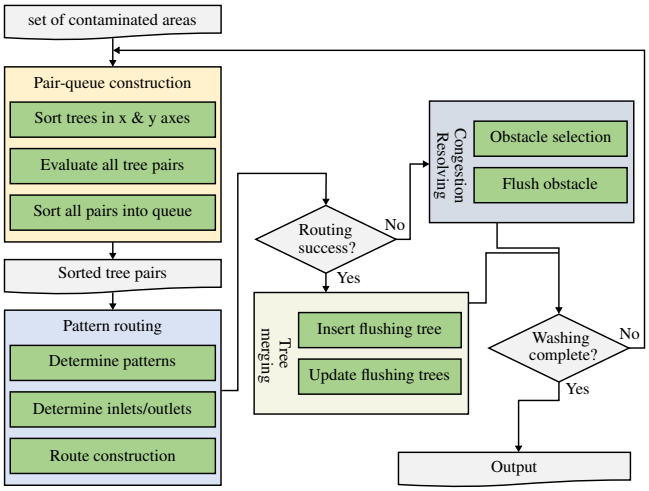


Fig. 6. Flushing tree construction flow.

following subsections.

A. Pair-Queue Construction

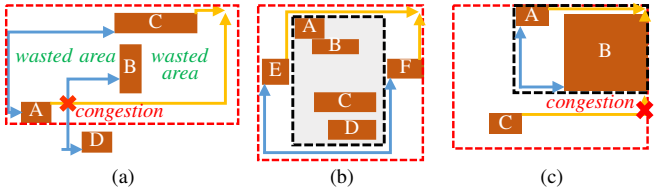
According to the schedule and location of the operations of a bioassay mapped onto different areas on the PMD by the user, the corresponding contaminated areas become known. These areas are the initial washing blocks, corresponding to the flushing trees shown in Fig. 5(a). These flushing trees are then merged recursively to collect washing blocks. However, if we merge flushing trees randomly, the following inefficient results may occur.

1) white-space wasting

If we merge two different flushing trees without considering the distance between them, lots of routable areas may be split and wasted. Moreover, by combining two remote trees, the lengths of washing paths would be prolonged so that the efficiency of routing is decreased. An example is shown in Fig. 7(a). In this case, block A and block C are bound first and block B and block D are bound afterwards. However, because the distance between A and C is too large, this pattern combination covers a chip area larger than necessary. Moreover, some washing blocks may be enclosed in such area and cannot be flushed. For example, block B is surrounded by the buffer flow paths so that no clean path can be constructed to wash it. This phenomenon is called congestion.

2) Gap Binding

After several rounds of binding trees, we may produce a massive flushing tree. If we do not consider the size problem during binding the flushing trees, the massive tree may occupy too many available areas in the chip. Assume that in Fig. 7(b) several washing blocks in the middle of the chip are collected into a massive flushing tree. The connection of block E and block F is thus prolonged because most of the available areas in the chip have been occupied. With the increasing length of the washing paths between block E and block F, the efficiency of routing is decreased.



(a) (b) (c)

Fig. 7. Unfavorable binding results. (a) White-spacing wasting. (b) Gap binding. (c) Source-sink blocking.

3) Source-Sink Blocking

During the process of binding flushing trees, some flushing trees may block all potential routing paths from the buffer source to the buffer sink. As shown in Fig. 7(c), after binding block A and block B, the buffer sink is completely blocked so that no path for block C can arrive at the buffer sink.

To deal with the problems above, we propose an effective method to choose the tree pairs for binding. In this method, we sort the flushing trees under different criteria respectively. One criterion is to sort them according to the x-axis coordinates of the inlets of the washing blocks. The other criterion is to sort them according to the y-axis coordinates of the inlets. From each sorted queue, we combine two close flushing trees sequentially to form a pair. For example, if we have a queue of sorted trees containing A, B, and C, the results of pairs are (A, B) and (B, C). By collecting these trees according to the x-axis and y-axis orders, we can ensure that the trees in the same pair can be bound in a reasonable distance.

The pairs in the queues are evaluated using the following cost function, which considers the distance and combination size of flushing trees in the same pair, as

$$\begin{aligned} Cost_{m,n} = \alpha \times \frac{S_{m,n} - S_{m,m} - S_{n,n}}{S_{m,n}} \\ + \beta \times \left| \frac{S_{m,m} - S_{n,n}}{max(S_{m,m}, S_{n,n})} \right| \\ + \gamma \times \frac{S_{m,n}}{ChipSize} \end{aligned} \quad (1)$$

where $Cost_{m,n}$ is the cost of a pair containing $Tree_m$ and $Tree_n$. $S_{m,m}$ and $S_{n,n}$ are the sizes of $Tree_m$ and $Tree_n$, respectively. $S_{m,n}$ is the area size determined by the inlet of $Tree_m$ and the outlet of $Tree_n$. α, β and γ are constants set by the user. The first term of the cost function considers white spaces after combination. If the amount $S_{m,n} - S_{m,m} - S_{n,n}$ is large, lots of areas would be wasted after combination. The second and last terms of the cost function consider the sizes of the flushing trees. To avoid building a massive tree, we combine the trees in a pair with similar sizes, as specified by the second term. The last term also helps prevent building a massive tree. Source-sink blocking can be resolved simply by determining the new inlet and new outlet of the combination in advance and discarding the pair whose combination would block all paths from the source to the sink.

B. Pattern Routing

The step above evaluates all pairs of flushing trees and sorts them into a priority queue according to the cost function. We

then sequentially choose the pairs with the least cost in the priority queue. To check if the current pair can be merged, we examine which pattern for binding is the most suitable for the pair. The pattern suitable for the selected pair can be determined by the relative position of the corresponding flushing trees. The situations of selecting different patterns are listed as follows:

- ***Merge in L shape:*** The pair can be bound in the L shape if one of the flushing trees is located at the top-left and another at the bottom-right.
- ***Merge in C shape:*** The pair can be bound in the C shape if they do not overlap with each other in the horizontal direction.
- ***Merge in U shape:*** The pair is can be bound in the U shape if they do not overlap with each other in the vertical direction.

When examining the flushing trees in a pair, we first check their relative positions complying with the rule of merging in the L shape. If it fails, the relative positions of this pair must meet one of the other two rules to be bound with the C shape or the U shape, because two blocks cannot overlap in the horizontal and vertical directions simultaneously.

After choosing the suitable pattern for a block pair, we determine the positions of the new inlet and the new outlet for this combination, as shown in Fig. 4(c)–(e). In the L shape, the new inlet is set at the position that is the closest point to the buffer source after combining. Different from the L shape, we keep a short distance for the new inlet in the C shape or the U shape from the original washing blocks to ensure that the buffer fluids are clean before being injected into these blocks. Without this short distance, the new inlet in Fig. 4(d) or Fig. 4(e) would be set at the original inlet of block A. Consequently, the buffer fluid to block B would be contaminated before reaching B. Similar to the inlet, the new outlet is set at the closest point to the buffer sink in the L shape and with a small offset in the C and U shapes.

After determining the locations of the new inlet and the new outlet, we apply L-shape routing as shown in Fig. 4(c) to connect the flushing trees inside the same pair. L-shape routing can guarantee that the routing result is connected with the shortest distance. If this routing succeeds, a new washing block is created. Otherwise, there should be some obstacles blocking the shortest paths. In this case, we continue to check the following pairs to verify whether a feasible routing can be found. If finally this is not possible, a special measure to resolve routing congestion is applied, as described in Section IV-D.

C. Tree Merging

After finding a washing pair with a feasible routing, this newly merged washing block corresponds to a new unconnected node in the flushing trees and all the previous washing blocks covered by the new block are removed to avoid duplicated washing. The flushing trees are then processed by repeating the previous steps. These iterative steps gradually reduce the number of the flushing trees until only one super block is left, which is thus the solution for washing all the contaminated areas on the PMD.

D. Congestion Resolving

If there is no feasible routing solution in all the pairs of washing blocks, a remedial measure is applied to release the occupied areas. In the congestion resolving stage, we choose a flushing tree that blocks the most available areas in the chip and wash it first. In order to choose a suitable tree, a cost function is applied as follows

$$Cost = \alpha \times \frac{|x_{inlet} - Width/2| + |y_{inlet} - Height/2|}{2} + \beta \times \frac{ChipSize}{(x_{inlet} - x_{outlet}) \times (y_{inlet} - y_{outlet})} \quad (2)$$

where x_{inlet} and y_{inlet} are the x-axis and y-axis coordinates of the inlet of a flushing tree. x_{outlet} and y_{outlet} are the coordinates of the outlet of the flushing tree. $Width$ and $Height$ are the width and height of the chip, respectively.

The first term of (2) considers the position of the flushing tree and indicates that the washing blocks located close to the center of the chip are resolved first to remove routing congestion. The second term of (2) considers the size of the flushing tree. If the flushing tree occupies a large area on the chip, it has a priority in congestion resolving.

V. Simulation Results

The proposed algorithm was implemented in the C++ programming language and tested with a 3.7 GHz CPU and 64GB memory. We used the benchmarks in [8] as bioassays executed on PMDs. The library containing different sizes of devices mapped on the PMD is summarized in Table I. These devices are used to execute the operations in the bioassays.

We compare our algorithm with two baseline solutions. The first one is Sequentially Washing, where the contaminated blocks are washed independently one after another. After choosing a feasible block, we directly wash it and release the occupied area. Because washing blocks are not collected together, this method is the most time-consuming one among all the solutions. The second baseline solution is Randomly Binding, where the contaminated blocks are bound together randomly. Therefore, many unfavorable combinations shown in Fig. 7(c) may appear and congestion resolving is applied very frequently in this solution.

Table II compares the washing time, CPU time, and the remedial numbers of the proposed algorithm, Randomly Binding and Sequential Washing, respectively. Block Number represents the number of contaminated areas in the chip. Washing time represents the overall time of washing, which is the most important criterion to evaluate the efficiency of washing. Remedial number represents how often congestion resolving needs to be applied. The relative ratios of these metrics are shown in the

TABLE I
DEVICE LIBRARY

Volume	4	6	8	8	10	10
Dimensions	2x2	2x3	2x4	3x3	2x5	3x4
Ratio	1:1	1:2	1:1, 1:3	1:1, 1:3	1:4, 2:3	1:4, 2:3

TABLE II
WASHING TIME, CPU TIME, AND REMEDIAL NUMBERS OF THE PROPOSED METHOD AND TWO BASELINE SOLUTIONS

Benchmark	Chip Size	Block Number	Proposed Algorithm			Randomly Binding			Sequential Washing		
			Washing Time(s)	CPU Time(s)	Remedial Number	Washing Time(s)	CPU Time(s)	Remedial Number	Washing Time(s)	CPU Time(s)	Remedial Number
PCR	20 x 20	7	2.5	0.0007	4	3	0.0007	5	3.5	0.0008	7
Mixing Tree (MT)	30 x 30	18	3.5	0.0029	6	5	0.0043	9	9	0.0071	18
Interpolating Dilution (PID)	40 x 40	39	12	0.014	23	15	0.019	29	19.5	0.02	39
Exponential Dilution (PED)	45 x 45	55	19	0.042	37	24	0.067	47	27.5	0.073	55
N. Average			1	1	1	1.286	1.36	1.32	1.761	1.69	1.985

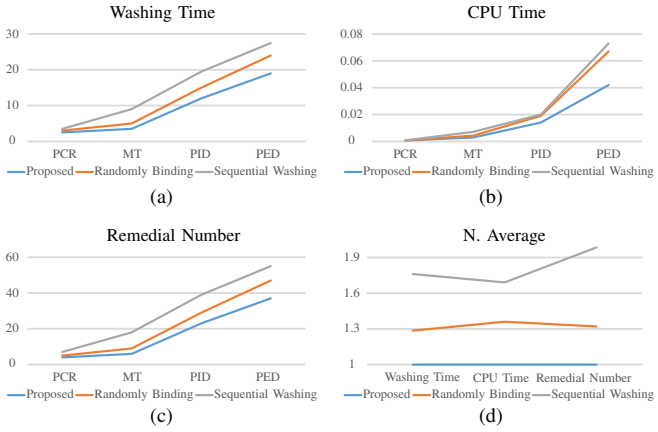


Fig. 8. Comparison of the proposed method and the baseline solutions. (a) Washing time. (b) CPU time. (c) Remedial Numbers. (d) N. Average.

row N. Average.

In the simulation, the diffusion times of the buffer flows were considered negligible compared with the propagation time of the flows along the washing paths. Therefore, this factor is ignored to simplify the simulation. Moreover, since the washing paths in the flushing operations are the shortest paths, meaning that the lengths of all paths are equal, we can reasonably assume that the time taken by each flushing operation is the same, which was set to 0.5 seconds in the experiments.

As shown in Table II, without collecting any blocks together, the remedial congestion resolving operation is applied frequently and lowers the washing efficiency significantly in Sequential Washing. Moreover, because the routing area in Sequential Washing is always larger than in the proposed algorithm, the CPU time to find feasible routing solutions in Sequential Washing is also larger than our method. Compared with Randomly Binding, our remedial numbers are smaller because the pairs of washing blocks are selected carefully for binding, which avoids the frequent occurrences of unfavorable combinations in Randomly Binding.

Fig. 8 shows the comparison among different methods for each benchmark. These evaluations show that the proposed algorithm outperforms other baseline solutions especially when the sizes of benchmarks are large.

VI. Conclusions

In this paper, we have proposed the first washing algorithm applying flushing operations in programmable microfluidic devices. In our algorithm, we efficiently collected the contaminated areas by constructing flushing trees. Moreover, given patterns were followed to merge flushing trees recursively to decrease the scale of routing significantly. Simulation results showed that the proposed algorithm can achieve the best washing time and CPU time compared with baseline solutions especially when the sizes of benchmarks are large.

Acknowledgment

The work of T.-Y. Ho was supported in part by the Technical University of Munich – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763. The work of B. Li and U. Schlichtmann was supported in part by Deutsche Forschungsgemeinschaft (DFG) through TUM International Graduate School of Science and Engineering (IGSSE).

References

[1] T. M. Squires and S. R. Quake, "Microfluidics: Fluid physics at the nanoliter scale," *Reviews of Modern Physics*, vol. 77, pp. 977–1026, October 2005.

[2] M. A. Unger, H.-P. Chou, T. Thorsen, A. Scherer, and S. R. Quake, "Monolithic microfabricated valves and pumps by multilayer soft lithography," *Science*, vol. 288, no. 5463, pp. 113–116, 2000.

[3] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T. Ho, and U. Schlichtmann, "Testing microfluidic fully programmable valve arrays (FPVAs)," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pp. 91–96.

[4] L. M. Fidalgo and S. J. Maerkl, "A software-programmable microfluidic device for automated biology," *Lab Chip*, vol. 11, pp. 1612–1619, 2011.

[5] J. Melin and S. R. Quake, "Microfluidic large-scale integration: The evolution of design rules for biological automation," *Annual Review of Biophysics and Biomolecular Structure*, vol. 36, no. 1, pp. 213–231, 2007.

[6] L. M. Fidalgo and S. J. Maerkl, "Fluid transportation in PMD," http://www.rsc.org/suppdata/lc/c0/c0lc00537a/videos2.mov, 2011.

[7] ——, "Mixing operation in PMD," http://www.rsc.org/suppdata/lc/c0/c0lc00537a/videos3.mov, 2011.

[8] T. Tseng, B. Li, M. Li, T. Ho, and U. Schlichtmann, "Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1981–1994, March 2016.

[9] G. Lai, C. Lin, and T. Ho, "Pump-aware flow routing algorithm for programmable microfluidic devices," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1405–1410.

[10] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille, "Close-to-optimal placement and routing for continuous-flow microfluidic biochips," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 530–535.

[11] Q. Wang, S. Zuo, H. Yao, T.-Y. 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.

[12] K. Hu, T. Ho, and K. Chakrabarty, "Wash optimization and analysis for cross-contamination removal under physical constraints in flow-based microfluidic biochips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 559–572, April 2016.