Optimal Scheduling of Biochemical Analyses on Digital Microfluidic Systems

Lingzhi Luo, *Student Member, IEEE*, and Srinivas Akella, *Member, IEEE*

***Abstract*—Digital microfluidic systems (DMFS) are an emerging class of lab-on-a-chip systems that manipulate individual droplets of chemicals on a planar array of electrodes. The biochemical analyses are performed by repeatedly moving, mixing, and splitting droplets on the electrodes. Mixers and storage units, composed of electrodes, are two important functional resources. Mixers perform droplet mixing and splitting operations, while storage units store droplets that have been produced for subsequent mixings. In this paper, we focus on minimizing the completion time of biochemical analyses by exploiting the binary tree representation of analyses to schedule mixing operations. Using pipelining, we overlap mixing operations with input and transportation operations. We find the lower bound of the mixing completion time based on the tree structure of input analyses, and calculate the minimum number of mixers M^{lb} required to achieve the lower bound. We present a scheduling algorithm for the case with a specified number of mixers M, and prove it is optimal to minimize the mixing completion time. We also analyze resource constraint issues for two extreme cases. For the case with just one mixer, we prove that all schedules that keep the mixer busy at all times result in the same mixing completion time and then design algorithms for scheduling and to minimize the number of storage units. For the case with zero storage units, we find the minimum number of mixers required. We extend our analyses and algorithms assuming identical mixing durations to the case of different mixing durations. Finally, we illustrate the benefits of our scheduling methods on an example of DNA polymerase chain reaction (PCR) analysis.**

***Note to Practitioners*—This paper presents scheduling techniques for DMFS, a new class of lab-on-a-chip devices. These chips execute chemical analyses by moving, mixing, and splitting droplets of different chemicals. Our primary goal is to automate and optimize the scheduling of droplet input, mixing, and output operations to minimize the completion time of DMFS applications. This paper presents new scheduling algorithms that are efficient, easy to implement, and proven to be optimal in completion time. The algorithms exploit the full binary tree structure of biochemical analyses to minimize the completion time. To develop cost-efficient biochips, we present algorithms that reduce the required resources (e.g., mixers, storage units) on DMFS chips to complete the biochemical analyses. Our scheduling algorithms when combined with droplet routing algorithms can create completely automated software controllers for DMFS.**

***Index Terms*—Digital microfluidic systems (DMFS), lab-on-a-chip, resource constraint, scheduling algorithm.**

Manuscript received August 25, 2008; accepted February 26, 2009. Date of publication July 29, 2010; date of current version January 07, 2011. This paper was recommended for publication by Associate Editor F. Arai and Editor D. Meldrum upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Award IIS-0093233, Award IIS-0541224, Award IIS-0713517, Award IIS-0730817, and CNS-0709099.

L. Luo is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: lingzhil@cs.cmu.edu).

S. Akella is with the Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223 USA (e-mail: s.akella@ieee.org).

Digital Object Identifier 10.1109/TASE.2010.2053201

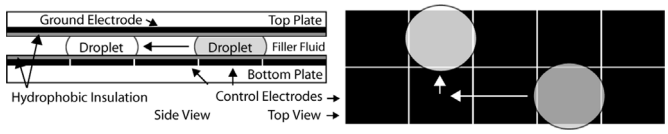


Fig. 1. Droplets on an electrowetting array (side and top views). The droplets are in a medium (usually oil or air) between two glass plates. The gray and white droplets represent the same droplet in initial and destination positions. A droplet moves to a neighboring electrode when that electrode is activated; the electrode is turned off when the droplet has completed its motion. Based on [6].

I. Introduction

LOW-COST, portable lab-on-a-chip systems capable of rapid automated biochemical analysis can impact a wide variety of applications including biological research, genetic analysis, point-of-care diagnostics, and biochemical sensing [1]–[4]. *Digital microfluidic systems* (DMFS) are an emerging class of lab-on-a-chip systems that manipulate discrete droplets [5], [6]. A digital microfluidic system manipulates individual droplets of chemicals on a planar array of electrodes by using electrowetting (Fig. 1) or dielectrophoresis. The chemical analysis is performed by repeatedly moving, mixing, and splitting droplets on the electrodes.

An important advantage of DMFS devices is their reconfigurability and flexibility in performing various biochemical analyses. We focus on microfluidic systems that manipulate droplets by electrowetting [7]. Droplets are 100–1000 nanoliters in volume, and have been moved at 12–25 cm/s on planar arrays of 0.15 cm wide electrodes [5], [8]. The ability to control discrete droplets on a planar array enables complex analysis operations to be performed in DMFS devices. For simple biochemical analysis operations, no special purpose devices are required aside from the array itself. Mixers and storage units are two important resource components consisting of electrodes. Mixers are used to perform mixing and splitting operations, while storage units are used to store droplets which have been produced for subsequent mixings (Fig. 2). Transportation paths are used to move droplets among different components (e.g., mixers and storage units). The array may additionally contain cells that can perform specialized operations, such as heating or optical sensing.

The completion time of biochemical analyses in batch mode is the time required to produce one droplet of final product. Our goal is to minimize the completion time in batch mode under resource constraints on DMFS. Practical benefits include time-efficient scheduling of multiple analyses, and cost-efficient design of DMFS biochips (since smaller chips are cheaper to fabricate). The completion time of a reaction depends on the

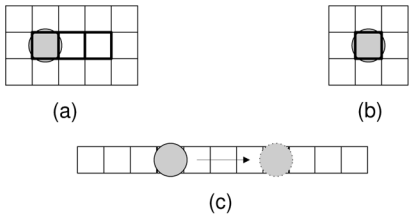


Fig. 2. A schematic of a mixer, storage unit, and transportation path. (a) A mixer with three electrodes for droplets to move. (b) A storage unit with 1 electrode for droplets to stay. (c) Transportation path for droplets.

reaction, the provided resources, and the algorithm to perform it. In this paper, we exploit the tree structure of biochemical analyses to perform scheduling and minimize the completion time. We introduce a full binary tree representation of the biochemical reactions for convenience in algorithm design and analysis. Using pipelining we overlap the mixing operations with input and transportation operations. We focus on scheduling of mixing operations to minimize the mixing completion time. We calculate the lower bound of a given reaction based on the tree structure. We also calculate the minimum number of mixers required to achieve minimum completion time, and present a greedy scheduling algorithm to minimize the completion time under mixer constraints. With the above algorithm, parallelism is fully exploited not only among different mixing steps but also among different kinds of operations. We analyze and design our algorithms initially assuming identical mixing duration and then generalize to the case with different mixing durations. For optimal scheduling with a given number of mixers, we reduce the problem with different mixing durations to an equivalent problem with identical mixing duration based on preemption. We next consider two extreme cases of resource constraints: with just one mixer and with zero storage units. In the first case, we prove all active schedules (i.e., that keep the mixer busy at all times) result in the same completion time, calculate the minimum number of storage units for the given analysis, and design a corresponding scheduling algorithm. In the second case, we compute the minimum number of mixers required. These results can be used to guide the selection of chips and layout design with a cost and space-efficient number of mixers and storage units. The complexity of algorithms described in this paper is linear in the number of nodes in the tree structure.

A preliminary version of this work appeared in [9].

II. Related Work

Scheduling Algorithms: Scheduling algorithms optimize the system performance by properly allocating tasks to resources. Kwok and Ahmad [10] studied the static scheduling of a program on a multiprocessor system to minimize the program completion time in parallel processing. Since the general problem is NP-complete, they compared 27 heuristic scheduling algorithms. In contrast to the general problem in parallel computing, the multiple-task reactions that we consider in DMFS have certain kinds of precedence. Ding *et al.* [11], Su and Chakrabarty [12] represent the DMFS reactions using data-flow directed graphs and consider the scheduling using integer linear programming (ILP). They solve the problem by general ILP solvers and heuristic algorithms without exploiting the full binary tree structure of DMFS reactions.

Design Automation: Su and Chakrabarty [12], [13] presented architecture-level synthesis and geometry synthesis of biochips. They used acyclic sequence graphs to represent the reactions and developed techniques in operation scheduling, resource binding, and module placement. In their papers, resource constraints are given in advance by the size of chips or the number of mixers and storage units. One of our goals is to develop guidelines for selecting and designing chips with the proper number of mixers and storage units for given biochemical analysis.

Routing: Minimizing the routing time and avoiding undesired droplet collisions are two important problems for routing algorithm design. Böhringer [14] modeled the routing problem in DMFS as a multirobot motion planning problem and used a prioritized A^* search algorithm to generate the optimal plan for droplets. Griffith and Akella [15], [16] presented a general-purpose DMFS and designed routing algorithms based on Dijkstra's algorithm. Su *et al.* [17] proposed a two-stage routing method to minimize the number of electrodes used for droplet routing, while considering the resource constraint. In our paper, the time when droplets are input is controlled based on the result of pipelining techniques applied. Thus, only those droplets just produced or immediately required will be transported so that the load of routing paths is reduced.

Layout Design: The purpose of layout design is to map the functional units such as mixers, storage units, and transport paths to the underlying hardware. Griffith and Akella [15] presented a semi-automated method to generate the array layout in terms of components. Su and Chakrabarty [18] developed an online reconfigurable technique to bypass faulty electrodes in the microfluidic biochips. In these papers, the chips for layout design are either given in advance or designed with some human input.

III. Binary Tree Model of Biochemical Analysis

The biochemical "analysis graph" provides a representation of the operations of DMFS. It is a directed graph, with an input node for each droplet type entering the system, an output node for each product droplet type leaving the system, and a mix node for each mixing operation performed in the system. (Each mix operation is followed by a splitting operation to maintain uniform droplet volume.) The nodes are connected based on the droplet types they require and produce, and the edges represent these droplet transport operations. For example, consider the PCR analysis graph shown in Fig. 3. The mixing operations during the DNA Polymerase Chain Reaction (PCR) analysis are described as follows:

$$A + B \rightarrow C; C + D \rightarrow E; E + F \rightarrow G;$$
$$H + I \rightarrow J; J + K \rightarrow L; L + M \rightarrow N; N + G \rightarrow O.$$

Here, $A+B \rightarrow C$ means reagents A and B are mixed to produce C. To model the dependencies among different mixing opera-

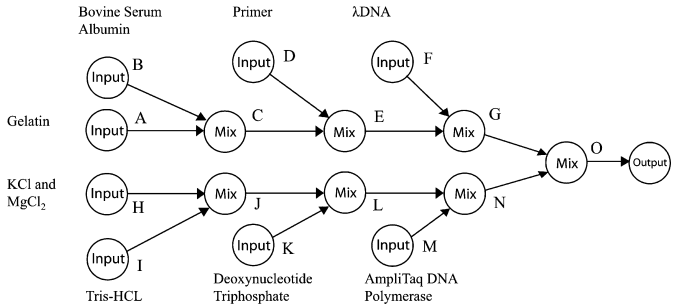


Fig. 3. PCR analysis graph. Input nodes are labeled with the reagents they introduce and alphabet labels for convenience. Mix nodes represent the mixing operations. The output node represents the final product.

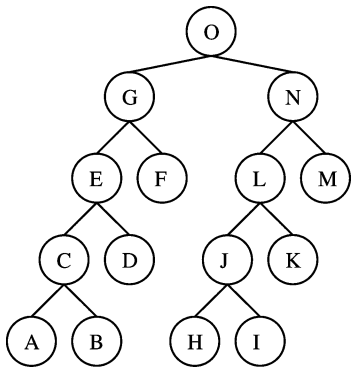


Fig. 4. Representation of PCR analysis by a full binary tree with depth 4.

tions, we introduce a full binary tree structure. A *full binary tree* is a binary tree in which every node is either a leaf node or it has two child nodes [19]. Given an analysis R, we construct a full binary tree T as follows. Every reagent in R is represented by a node in T. Source reagents, which can be directly fetched from the reservoirs, are represented by leaf nodes in T, and intermediate reagents, which are produced by mixing, are represented by parent nodes of those nodes from which they can be produced. So, the analysis R is represented by T, where the final product of R is represented by the root node of T. The representation of the PCR analysis of Fig. 3 by a full binary tree is shown in Fig. 4.

IV. Scheduling Algorithm Design

For a general biochemical analysis, there are five kinds of basic operations: *input*, *transport*, *mix* (including split), *store*, and *output*. The execution order of these operations should satisfy the following rules.

1) *Operation precedence rule*: The precedence order of related operations for a mixing $A + B \rightarrow C$ is: *Input* A and $B \prec \mathrm{Transport}A$ and B to a mixer $\prec \mathrm{Mix}A$ and B for $C \prec \mathrm{Output/Store/Transport}C$. ($\prec$ represents a precedence relation.)
2) *Mixing precedence rule*: In the binary tree model, the mixing operation for a parent node should occur after the mixing operations for its child nodes.
3) *Resource constraint rule*: At any time the utilized resources (e.g., mixers and storage units) should not exceed the resources of the biochip.

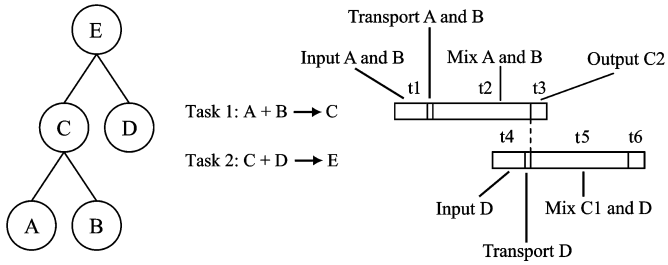


Fig. 5. Application of pipelining to an analysis on DMFS. After mixing operations (including splitting), two destination droplets are produced, labeled with 1 and 2, respectively. The output operations are used to dispense one waste droplet (labeled with 2) outside the system. The second mixing task is also performed in the same mixer, so droplet C1 does not need to be transported for the second mixing task.

Our scheduling design is composed of two parts: pipelining and mix scheduling. Pipelining overlaps mixing operations with other operations, while satisfying the operation precedence rule. Scheduling of mixing operations reduces the total mixing time, while satisfying the mixing precedence rule. Both pipelining and mix scheduling should satisfy the resource constraint rule.

A. Pipelining

Pipelining is a scheduling method that divides a task into several subtasks and performs different subtasks of multiple tasks in different function units simultaneously to reduce the overall completion time of all tasks [20]. Using pipelining, the completion time of a single task will depend on the subtask that takes the longest time.

For biochemical analyses on a DMFS, the procedure of obtaining a product droplet can be divided into several subtasks: input source droplets, transport droplets, mix (and split) droplets, and output waste droplets. Among these subtasks, droplet mixing takes the longest time. The mixing duration depends on the mixer size, droplet motion pattern, and the chemicals to be mixed [7]. To simplify our analysis, we initially assume all mixing operations have an identical duration. Fig. 5 shows an example of applying pipelining to a part of PCR analysis.

Suppose $t_{\mathrm{completion}}$ is the completion time, $T_{\mathrm{input}}, T_{\mathrm{output}}$, and T_{mix} are the durations for one input, output, and mixing operation, respectively, $t_{\mathrm{mix}}^{\mathrm{total}}$ is the total time for mixing, and $t_{\mathrm{transport}}$ is the time for transportation operations

$$t_{\mathrm{completion}} = T_{\mathrm{input}} + t_{\mathrm{mix}}^{\mathrm{total}} + t_{\mathrm{transport}} + T_{\mathrm{output}} \quad (1)$$

where $t_{\mathrm{mix}}^{\mathrm{total}}$ equals $\sum T_{\mathrm{mix}}$.

Since the transportation time is negligible compared to the mixing time [5], [8], we ignore the transportation time in subsequent scheduling. The approximate completion time is

$$t_{\mathrm{completion}} \approx T_{\mathrm{input}} + t_{\mathrm{mix}}^{\mathrm{total}} + T_{\mathrm{output}}. \quad (2)$$

Since $T_{\mathrm{input}}, T_{\mathrm{output}}$, and T_{mix} are constants, we only consider optimization of mixing completion time $t_{\mathrm{mix}}^{\mathrm{total}}$ below.

B. Scheduling Mixing Operations

The goal of mixer scheduling is to minimize $t_{\text{mix}}^{\text{total}}$ using the given number of mixers. $t_{\text{mix}}^{\text{total}}$ depends on three factors.

1) The tree structure of the biochemical analysis.
2) The number of mixers provided.
3) The scheduling algorithm.

In the following sections, we will answer these central questions.

1) What is the lower bound of the mixing completion time for a biochemical analysis?
2) What is the minimum number of mixers to achieve the lower bound on the mixing completion time?
3) What is the optimal scheduling algorithm with a limited number of mixers for an arbitrary analysis?
4) What if we have only one mixer or zero storage units?

We address the first two questions in Sections IV-C and IV-D, and the last two questions in Sections V and VI.

C. Lower Bound of Mixing Completion Time

1) Identical Mixing Durations: Suppose T is a full binary tree that represents the biochemical analysis R, the depth of T is D, and the number of nodes at level i is N_i, $i = 0, \ldots D$. (The root node of T is at level 0 and the deepest leaf nodes are at level D.) Here, the mixing duration T_{mix} is defined as the duration from the time when the second reagent arrives at the mixer entrance to the time when the mixed product exits the mixer.

Lemma 1: The lower bound $T_{\text{lowerbound}}$ of $t_{\text{mix}}^{\text{total}}$ is the sum of mixing durations along the deepest branch of T

$$T_{\text{lowerbound}} = D \cdot T_{\text{mix}}. \tag{3}$$

Proof: First, we show that all mixing operations can be completed in $D \cdot T_{\text{mix}}$ time with $\max_{i=1}^{D} (N_i)/(2)$ mixers. We perform the mixing operations from the deepest nodes to the highest nodes until the root node at level 0. Since the number of mixers $\max_{i=1}^{D} (N_i)/(2) \geq (N_i)/(2)$, we can schedule all the mixing operations at level i in a mixing duration T_{mix}. After the mixing operation on level 1, we get the root node. In this case, $t_{\text{mix}}^{\text{total}} = D \cdot T_{\text{mix}}$. So, $T_{\text{lowerbound}} \leq D \cdot T_{\text{mix}}$.

Second, we show that $T_{\text{lowerbound}} \geq D \cdot T_{\text{mix}}$. Suppose we can perform all mixing operations in less time, that is, $T_{\text{lowerbound}} < D \cdot T_{\text{mix}}$. Then according to the *pigeonhole principle*, there must be at least two mixing operations along the deepest branch of the tree performed in the same time slot, which contradicts the mixing precedence rule. ■

2) Extension to Different Mixing Durations: We now consider the case where each mixing operation may have a different mixing duration. Let B_{dpst} denote the branch with the largest sum of mixing time durations starting from the root node, b_i denote the ith node along B_{dpst}, and $T_{\text{mix}}^{b_i}$ denote the duration of the mixing operation to obtain b_i.

Lemma 2: The lower bound $T_{\text{lowerbound}}$ of $t_{\text{mix}}^{\text{total}}$ is the sum of mixing durations along the branch B_{dpst} with the biggest sum of mixing time durations

$$T_{\text{lowerbound}} = \sum_{b_i \in B_{dpst}} T_{\text{mix}}^{b_i}. \tag{4}$$

Proof: The proof of this lemma is similar to the proof of Lemma 1. First, given sufficient number of mixers and storage units, mixing operations along B_{dpst} can be performed consecutively without delay from completing other branches. In that case, $t_{\text{mix}}^{\text{total}} = \sum_{b_i \in B_{dpst}} T_{\text{mix}}^{b_i}$. Second, considering the *mixing precedence rule*, the mixing operations along B_{dpst} cannot be performed in parallel. So, $T_{\text{lowerbound}} = \sum_{b_i \in B_{dpst}} T_{\text{mix}}^{b_i}$. ■

We design Algorithm 1 to compute $T_{\text{lowerbound}}$, and B_{dpst} can be traced by following the "*T.root.next*" pointers stored by the algorithm. Algorithm 1 is designed based on two steps. First, base step $T_{\text{lowerbound}} = 0$ for a leaf node. Second, we have the induction step. Since the left and the right branches can be performed in parallel, the lower bound of completing both branches is the larger of their lower bounds. $T_{lb}^{T} = \max(T_{lb}^{T.left}, T_{lb}^{T.right}) + T_{\text{mix}}^{T.root}$. Due to the mixing precedence rule, the mixing operation to obtain $T.root$ should be performed after completing both left and right branches. So, we get the induction step.

Algorithm 1: Lower-Bound

Input: T// The full binary tree
Output: LB// Lower bound
if $T.root$ is a leaf node **then**
  $T.root.lb = 0$
  $T.root.next = null$
else
  $Left = \text{Lower-Bound}(T.left)$
  $Right = \text{Lower-Bound}(T.right)$
  $T.root.lb = \max(Left, Right) + T_{\text{mix}}^{T.root}$
  if $Left > Right$ **then**
    $T.root.next = T.left.root$
  else
    $T.root.next = T.right.root$
  end if
end if
return $T.root.lb$

D. Minimum Number of Mixers M^{lb} to Achieve the Lower Bound of Mixing Completion Time

To compute the minimum number of mixers to achieve the lower bound of mixing completion time, we introduce a lemma first. If we remove the reagent nodes of performed mixing operations after each time slot, the scheduling of mixing operations can be represented by a sequence of full binary trees. Let Tree_i

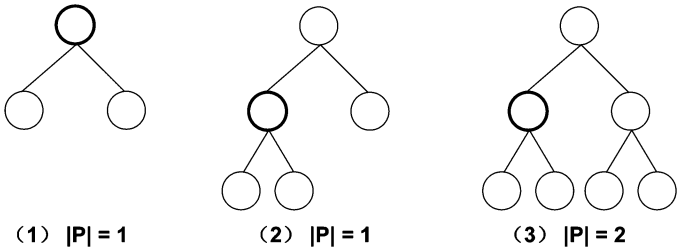


Fig. 6. Classification of mixing operations (shown in bold). Case 1: Mixing operation is not followed by subsequent mixing operations. Case 2: Mixing operation's parent's other reagent is a leaf node. Case 3: Mixing operation's parent's other reagent results from a mixing operation.

denote the full binary tree after the ith mixing duration, T_n denote the number of mixing durations for the final product. So, Tree_0 is the initial tree, and Tree_{T_n} is the root node.

Lemma 3: The number of mixing operations that can be performed for Tree_i, $i = 0, \ldots, T_n$, will change in nonincreasing order.

Proof: A mixing operation in Tree_i can be performed only when both reagents for that mixing are leaf nodes. Let P denote the set of such mixing operations. Each mixing operation belongs to one of three cases, based on its parent mixing operation, as shown in Fig. 6.

The second case will not change the size of P, while the first and third case will decrease the size of P by 1. ∎

Next, we calculate the minimum number of mixers required to achieve the lower bound of mixing completion time. We initially assume identical mixing durations T_{mix}. The sufficient and necessary condition is that the mixing operations along the deepest branch can be performed one by one in each mixing duration T_{mix}.

Let M^{lb} denote the minimum number of mixers to achieve the lower bound of mixing completion time, (i, j) denote the jth node in the ith level, $M_{(i,j)}$ denote the cumulative number of mixing operations to (i, j) from its leaf nodes, and N_i denote the number of nodes on the ith level.

Theorem 1: To achieve the lower bound of mixing completion time, the sufficient and necessary condition is

$$\sum_{j=1}^{N_i} M_{(i,j)} \leq M^{lb} \cdot (D-i) \quad \text{for all } i \in \{0, 1, \ldots, D-1\} \quad (5)$$

where

$$M_{(i,j)} = \begin{cases} 0 & \text{if } (i,j) \text{ is a leaf node;} \\ M_{(i+1,j_l)} + M_{(i+1,j_r)} + 1 & \text{if } (i+1,j_l) \text{ and } (i+1,j_r) \text{ are left and right child nodes of } (i,j) \end{cases}$$

That is,

$$M^{lb} = \max_{i=0}^{D-1} \left(\left\lceil \frac{\sum_{j=1}^{N_i} M_{(i,j)}}{D-i} \right\rceil \right). \quad (6)$$

Proof: When the deepest branch is processed up to the ith level, the required number of mixing operations is $\sum_{j=1}^{N_i} M_{(i,j)}$ and the number of mixing operations that can be performed by M^{lb} mixers during the elapsed $D-i$ time slots is $M^{lb} \cdot (D-i)$. So, the necessary condition is that (5) holds true.

Next, we show that (5) is also the sufficient condition for $t_{\mathrm{mix}}^{\mathrm{total}} = T_{\mathrm{lowerbound}}$ by proving that the number of mixers $M_z^{lb} = \max_{i=z}^{D-1} (\lceil (\sum_{j=1}^{N_i} M_{(i,j)})/(D-i) \rceil)$ can guarantee that nodes in level i can be produced before the end of time slot $D-i$ for all $i = D-1, \ldots, z$.

Base Step: $z = D-1$. $M_{D-1}^{lb} = \sum_{j=1}^{N_{D-1}} M_{(D-1,j)}$ is the number of mixing operations to produce all nodes at level $D-1$. The conclusion holds true.

Induction Step: For some level v, suppose that all mixing operations for nodes in levels $l > v$ can be performed using $M_{v+1}^{lb} = \max_{i=v+1}^{D-1} (\lceil (\sum_{j=1}^{N_i} M_{(i,j)})/(D-i) \rceil)$ mixers. $M_v^{lb} = \max_{i=v}^{D-1} (\lceil (\sum_{j=1}^{N_i} M_{(i,j)})/(D-i) \rceil)$. $M_v^{lb} \geq M_{v+1}^{lb}$. So, all mixing operations for nodes in levels $l > v$ can be performed using M_v^{lb} mixers. If in some time slot $D-l$, there are less number of mixing operations than M_v^{lb} that can be performed, according to Lemma 3, the situation will continue to time slot $D-v$. So, all mixing operations for nodes in all levels will be performed. Otherwise, if in all time slots from 1 to $D-v$, there are enough mixing operations to be performed, all mixing operations can also be performed in this case since $M_v^{lb} = \max_{i=v}^{D-1} (\lceil (\sum_{j=1}^{N_i} M_{(i,j)})/(D-i) \rceil) \geq (\lceil (\sum_{j=1}^{N_v} M_{(v,j)})/(D-v) \rceil)$.

So, the number of mixers $M_z^{lb} = \max_{i=z}^{D-1} (\lceil (\sum_{j=1}^{N_i} M_{(i,j)})/(D-i) \rceil)$ can guarantee that the mixing operations for nodes in level i can be performed before the end of time slot $D-i$ for all $i = D-1, \ldots, z$. So, the root node, which is in level 0, can be produced at the end of time slot D. So, the lower bound of mixing completion time is achieved using M^{lb} mixers. ∎

Using (6), we can directly compute the minimum number of mixers to achieve the lower bound of mixing completion time. However, this result only applies to the case with identical mixing duration. For the case with different mixing durations, we do not have an equation to directly compute the number. However, we can use binary search on the number of mixers until the optimal mixing schedule (see next section) achieves the lower bound of mixing completion time.

V. Optimal Mix Scheduling With Given Number of Mixers

A. Identical Mixing Durations

We now present a mix scheduling algorithm (Algorithm 2) for the case with a given number of mixers M. This is basically a greedy algorithm performing mixing operations from bottom to top.

- First perform the mixing operations at the deepest level and then proceed upwards.
- At each stage, perform as many mixing operations as possible using the given number of mixers M.

(This scheduling is in fact the critical path (CP) rule for parallel scheduling [21].)

Algorithm 2: Optimal-Scheduling-With-M-Mixers

Input: T, M*// The binary tree and the number of mixers*
Output: F *// Schedule*
$i = 1$ *// time slot index*
$F = \emptyset$
$C = \emptyset$ *// set of mixings that can be performed currently*
while $T \neq$ a tree with just a root node **do**
  Find pairs of leaf nodes with the same parent nodes in T *// identify mixing operations ready to be performed*
  Store the parent nodes in C, ordered from deepest level upwards
  if $|C| < M$ **then**
    $F = F \bigcup \{$Schedule all mixing operations in C in the time slot $i\}$
  else
    $F = F \bigcup \{$Schedule the first M mixing operations of C in the time slot i$\}$
  end if
  Remove from T leaf nodes involving the mixing operations added to F and remove their parent nodes from C
  $i = i + 1$
end while
return F

Theorem 2: Algorithm 2 is the optimal scheduling algorithm to minimize mixing completion time with M mixers.

Proof: Suppose algorithm O is optimal with M mixers to minimize mixing completion time and T_O, T_Q are the number of mixing durations of O and our algorithm, respectively.
First consider Algorithm 2 described above. Let i be the time slot when our algorithm finally (that is, last) encounters the condition that the number of mixings of the deepest-level nodes in C is larger than M. Let d denote the depth of T at the beginning of time slot i. According to Lemma 3 and the bottom-to-top execution order of Algorithm 2, $|C| > M$ was always satisfied before the ith time slot and no node at level smaller than d was mixed. Also, for every subsequent time slot, the algorithm will finish mixing operations of all nodes at subsequent level. That is, $T_Q = i + d$.
Suppose U is the number of mixing operations which need to be performed for nodes at level d or deeper. Let T_1 be the number of time slots spent mixing all nodes at level d or deeper by O, and T_2 be the number of time slots spent mixing all nodes at levels smaller than d by O. So, $T_O = T_1 + T_2$.
At the end of time slot i, our algorithm performed fewer mixing operations than U using M mixers, so $U > M \cdot i$, $T_1 \geq U/M \geq i + 1$. According to Lemma 1, $T_2 \geq d - 1$. Thus, $T_O = T_1 + T_2 \geq i + d$.
So, $T_O \geq T_Q$. That is, our algorithm is optimal to minimize the mixing completion time with M mixers. ■

B. Extension to Different Mixing Durations

We first consider extending the previous algorithm, and design Algorithm 3. The basic idea is, when selecting the mixing operations from the remaining tree, we just implement the previous algorithm as if all mixing durations were the same; when performing mixing operations, we use the real mixing durations.

Algorithm 3: Schedule-Diff-Mixing-Durations-With-M-Mixers

Input: T, M*// The binary tree and the number of mixers*
Output: F *// Schedule*
$t = 0$ *// present time*
$F = \emptyset$
$C = \emptyset$ *// set of mixings that can be performed currently*
while $T \neq$ a tree with just a root node **do**
  Find pairs of leaf nodes with the same parent nodes in T *// identify mixing operations ready to be performed*
  Store the parent nodes in C, ordered from deepest level upwards
  if $t == 0$ **then**
    if $|C| < M$ **then**
      $F = F \bigcup \{$Schedule all mixing operations in C at time $t\}$
    else
      $F = F \bigcup \{$Schedule the first M mixing operations of C at time t$\}$
    end if
  end if
  Find the first produced node i
  $t = t + T^i_{\text{mix}}$
  Remove from T leaf nodes involving the mixing operations added to F and remove their parent nodes from C
  if $C \neq \emptyset$ **then**
    $F = F \bigcup \{$Schedule the first mixing operation in C at time $t\}$
  end if
end while
return F

Algorithm 3 is the direct extension of Algorithm 2 taking into account the different mixing durations. However, it is not the optimal algorithm. Consider the example in Fig. 7. Algorithm 3 will select the mixing operation with the deepest level first,

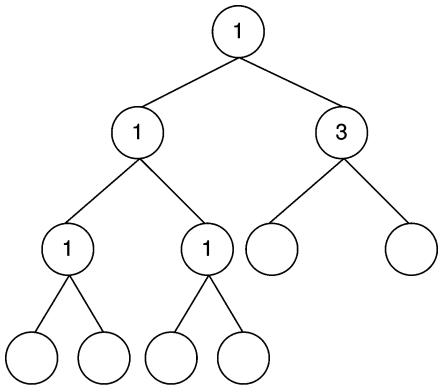


Fig. 7. A counterexample to the optimality of Algorithm 3 with two mixers. The number inside nonleaf node i is the remaining mixing time to produce i (i.e., $T^i_{\rm mix} - t_i$, where t_i is the time a mixer has been used to produce i). Note that there are no numbers inside leaf nodes since they have been produced.

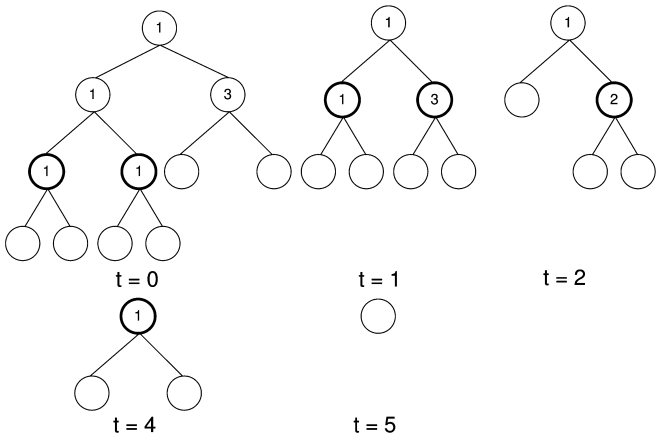


Fig. 8. The schedule by Algorithm 3 for the chemical analysis tree of Fig. 7 with two mixers. Nodes selected for mixing are shown bold. t denotes the elapsed time from the start of the mixing. A series of trees T updated in Algorithm 3 are shown with time labels t. The completion time using Algorithm 3 is five time slots.

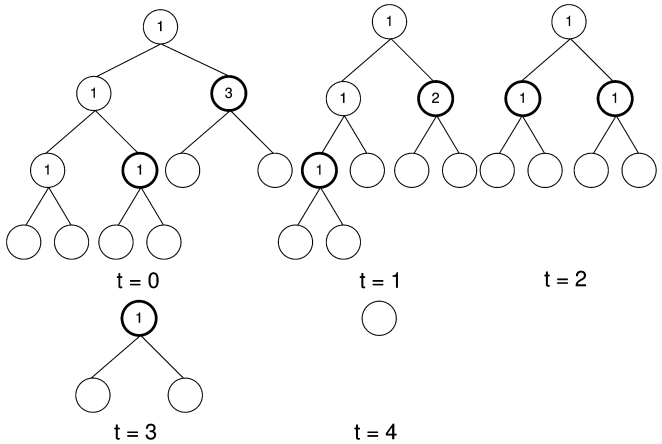


Fig. 9. The optimal schedule for the biochemical analysis tree in Fig. 7. A series of trees T updated in the optimal schedule are shown with time labels t. The completion time for the optimal schedule is four time slots.

as shown in Fig. 8. However, the optimal solution is shown in Fig. 9, where the lower bound is achieved.

Since the time durations are different, we do not use the concept of level for scheduling. Instead we use the sum of mixing durations, defined as *length*, along the branch from nodes to the root node.

The basic idea of Algorithm 4 is as follows:

1) First perform the mixing operations with the greatest length.
2) At each stage, perform as many mixing operations as possible using the given number of mixers.

The optimal solution shown in Fig. 9 can be achieved using Algorithm 4.

Algorithm 4: Scheduling-Diff-Mixing-Durations-Length

Input: T, M *// The binary tree and the number of mixers*

Output: $F, t^{\rm total}_{\rm mix}$ *// Schedule and mixing completion time*

$t = 0$ *// current time*

$F = \emptyset$

Traverse the tree to compute the lengths of nodes

$C = \emptyset$ *// set of mixings that can be performed currently*

while $T \neq$ a tree with just a root node **do**

Find pairs of leaf nodes with the same parent nodes in T *// identify mixing operations ready to be performed*

Store the parent nodes in C, in the decreasing order of their length

if $t == 0$ **then**

if $|C| < M$ **then**

$F = F \bigcup \{$Schedule all mixing operations in C at time t$\}$

else

$F = F \bigcup \{$Schedule the first M mixing operations of C at time t$\}$

end if

end if

Find the first produced node i

$t = t + T^i_{\rm mix}$

Remove from T leaf nodes involving the performed mixing operations and remove their parent nodes from C

if $C \neq \emptyset$ **then**

$F = F \bigcup \{$Schedule the first mixing operation in C at time t$\}$

end if

end while

$t^{\rm total}_{\rm mix} = t + T^{T.root}_{\rm mix}$

return $F, t^{\rm total}_{\rm mix}$

However, it is not the optimal algorithm either. Consider another example in Fig. 10. Algorithm 4 will select the mixing operation with the greatest length first as shown in Fig. 11. However the

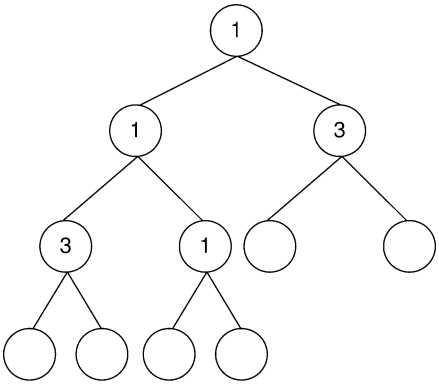


Fig. 10. A counterexample to the optimality of Algorithm 4 with two mixers. The numbers inside nonleaf nodes are the remaining mixing times to produce the nodes. Note that there are no numbers inside leaf nodes since they have been produced.

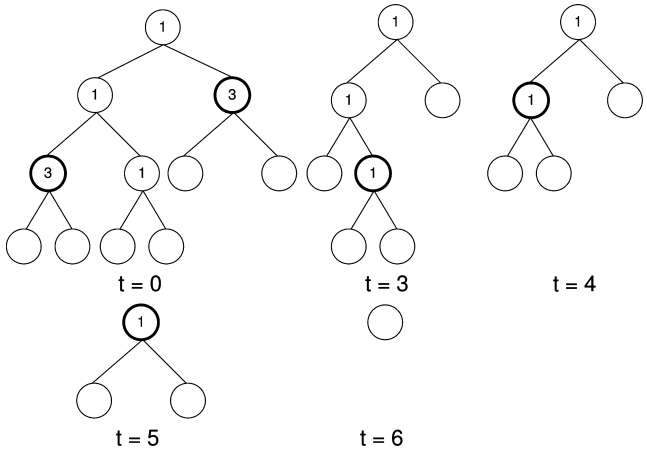


Fig. 11. Schedule by Algorithm 4 for the biochemical analysis tree of Fig. 10. t denotes the elapsed time from the start of the mixing. The numbers inside nonleaf nodes are the remaining mixing times to produce the nodes. Note that there are no numbers inside leaf nodes since they have been produced. A series of trees T updated in Algorithm 4 are shown with time labels t. The completion time using Algorithm 4 is six time slots.

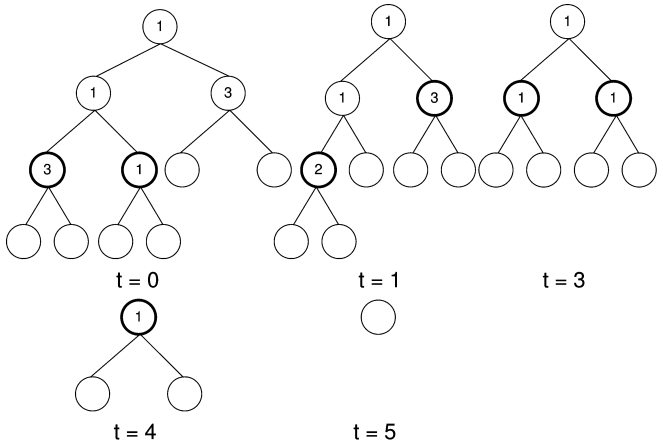


Fig. 12. The optimal schedule for the biochemical analysis tree in Fig. 10. A series of trees T updated in the optimal schedule are shown with time labels. The completion time for the optimal schedule is five time slots.

optimal solution which achieves the lower bound is shown in Fig. 12.

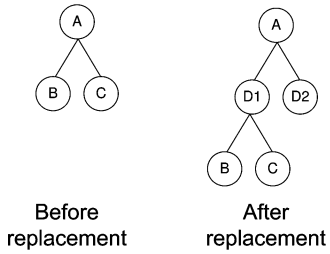


Fig. 13. Reducing the problem with different mixing durations to the problem with identical mixing durations by allowing preemptions in discrete time. $T^A_{\text{mix}} = 2$. Using preemptions, a tree with different mixing durations for its mixing operations can be replaced by another tree with unit mixing duration for all mixing operations ($T^A_{\text{mix}} = T^{D1}_{\text{mix}} = 1$). Nodes D_1 and D_2 are intermediate nodes of mixing and splitting nodes B and C, whose mixing operation to produce A is preempted. The connections of nodes A, B, C to other outer nodes should be maintained as before.

So, Algorithms 3 and 4 cannot guarantee the optimality of their solutions. Here, we apply the concept of preemption [21] and thus get the optimal completion time schedule based on preemption. In the context of DMFS, preemptions imply that a mixing operation can be interrupted at any time and when a preempted mixing operation resumes, it can be completed in its remaining processing time. By using preemptions, problems with different mixing durations can be reduced to equivalent problems with the same mixing duration; the structure of trees will change accordingly. We can then use Algorithm 2 to get the optimal schedule for problems with different mixing durations.

First, suppose the mixing durations are integers and the scheduler can preempt any mixer at integer times. (This can be relaxed later.) Due to the application of preemption, we can reduce this problem to an equivalent problem with identical mixing durations: For each mixing operation with duration T_{mix} greater than one, we can replace it with T_{mix} unit-duration mixing operations, as shown in Fig. 13. After this replacement step, the tree structure is changed and the problem with different mixing durations is reduced to an equivalent problem with identical mixing durations.

So, we can solve the equivalent problem with identical mixing durations using Algorithm 2 and get schedule F. Construct the optimal schedule for the problem with different mixing durations exactly the same as F. Note that if a storage unit is required for the new nodes introduced to a mixing operation by the replacement (e.g., nodes $D1$ and $D2$ in Fig. 13), then the mixing operation is interrupted when performed and its mixer is preempted at that time. Also, if we increase the resolution of mixing durations and reduce one unit of mixing duration to an arbitrarily small value ϵ, the problem intrinsically remains the same since the relative lengths of mixing durations do not change. By making ϵ sufficiently small, the above method can be applied to noninteger mixing times as well. The complexity of the algorithm is linear in the number of nodes in the modified tree structure.

Also, after replacement of mixing operations, the minimum number of mixers to achieve the lower bound can be calculated using Theorem 1.

VI. Resource Constraint Analysis for Two Extreme Cases

The resource constraints come from two aspects: the number of mixers and the number of storage units. The former limits how many mixing operations we can perform in a time slot. The latter limits the number of droplets that can be stored for mixing after having been produced.

A. Case 1: Scheduling With Only One Mixer

1) Identical Mixing Durations: At least one mixer is required to be able to perform any mixing operations. Let Q denote the number of mixing operations to perform for the biochemical analysis.

Lemma 4: With one mixer, all scheduling methods that keep the mixer busy will have the same $t_{\text{mix}}^{\text{total}}$

$$t_{\text{mix}}^{\text{total}} = Q \cdot T_{\text{mix}}. \tag{7}$$

Proof: No matter which scheduling method we use, only one mixing operation can be performed in each time slot due to the mixer resource constraint. As long as the mixer keeps busy, the mixing completion time is the sum of durations of all mixing operations. $t_{\text{mix}}^{\text{total}} = Q \cdot T_{\text{mix}}$. ■

Given a reaction tree T, we design Algorithm 5 to compute the minimum number of storage units, and Algorithm 6 to generate the corresponding schedule. Suppose $T.root$ is T's root node, $T.left$ and $T.right$ are the left and right subtrees of T. Since the tree is a full binary tree, either $T.root$ is a leaf node, or both $T.left$ and $T.right$ are non-null.

Algorithm 5: Min-Storage-Units

Input: T *// the binary tree*
Output: N *// the number of required storage units*
if $T.root$ is a leaf node **then**
  $T.value = 0$
else
  $Left = \text{Min-Storage-Units}(T.left)$
  $Right = \text{Min-Storage-Units}(T.right)$
  if $Left = Right$ **then**
    $T.value = Left + 1$
  else
    $T.value = \max(Left, Right)$
  end if
end if
if T is the initial input tree and $T.value \neq 0$ **then**
  $N = T.value - 1$
else
  $N = T.value$
end if
return N

Algorithm 6: One-Mixer-Scheduling

Input: T, i *// the binary tree and the time slot index (1 for the initial tree)*
Output: F *// Schedule, global variable initialized as* $\emptyset$
  if $T.value = 0$ **then**
  return
else
  if $T.left.value > T.right.value$ **then**
    One-Mixer-Scheduling$(T.left, i)$
    One-Mixer-Scheduling$(T.right, i)$
  else
    One-Mixer-Scheduling$(T.right, i)$
    One-Mixer-Scheduling$(T.left, i)$
  end if
end if
$F = F \bigcup \{\text{Schedule mixing to get } T.root \text{ in time slot } i\}$
$i = i + 1$
return F

Theorem 3: The minimum number of storage units for a reaction using one mixer is computed by Algorithm 5; the corresponding schedule in the order of execution is output by Algorithm 6.

Proof: Suppose $S(T)$ is the minimum number of storage units for the reaction represented by tree T using one mixer. First, we prove the first half of the theorem. If $T.root$ is a leaf node or $T.value = 1$ (the child nodes are leaf nodes), $S(T) = 0$. (Base step.)

If $T.root$ is not a leaf node, $S(T) \geq \max(S(T.left), S(T.right))$.

If $S(T.left) \neq S(T.right)$, without loss of generality assume $S(T.left) > S(T.right)$. Schedule as follows: first, perform all mixing operations in the left subtree, when we need $S(T.left)$ storage units; then perform those in the right subtree, when we need $S(T.right)$ storage units for the nodes in the right subtree and one for the $T.left$ node. So, $S(T) = \max(S(T.left), S(T.right))$.

If $S(T.left) = S(T.right)$, we show that $S(T) = S(T.left) + 1$. Perform all mixing operations in the left subtree, and then the right subtree. Here, we need $S(T.left) + 1$ storage units. So, $S(T) \leq S(T.left) + 1$. If $S(T) < S(T.left) + 1$, then two conditions should be satisfied: first, in the time slot when $S(T.left)$ storage units are used for the left subtree, no mixing operations in the right subtree have been performed; second, in the time slot when $S(T.right)$ storage units are used for the right subtree, no mixing operations in the left subtree have been performed. Obviously, they cannot be satisfied at the same time. By contradiction, we conclude $S(T) = S(T.left) + 1$. The induction step is also correct. So, N returned by the algorithm equals $S(T)$, the optimal value.

We now show that the second half of the theorem is correct. Since Algorithm 6 outputs the mixing operations in child-node-first order, the mixing precedence rule is satisfied. Also, the algorithm traces back through $T.value$ for all subtrees using the recurrence in Algorithm 5, so it outputs the corresponding scheduling results. ∎

2) Extension to Different Mixing Durations: The two conclusions above still hold true: 1) With one mixer, the total completion time is still the sum of all mixing durations as long as we keep the mixer busy all the time. (The proof of this conclusion is exactly the same as the case with an identical time duration.) 2) Algorithm 5 still computes the minimum number of storage units and Algorithm 6 still returns the corresponding schedule. With one mixer, the structure of the tree decides the number of required storage units. Since the structure is independent of the mixing durations, the minimum number of storage units and corresponding schedule will not change.

B. Case 2: Scheduling With Zero Storage Units

1) Identical Mixing Durations:

Theorem 4: The sufficient and necessary condition for performing all the mixing operations without storage units is to use $\max_{i=0}^{D}(N_i)/(2)$ mixers, and the completion time in this case is the lower bound $D \cdot T_{\text{mix}}$.

Proof: Actually, we have considered the sufficient condition when discussing the lower bound of mixing completion time in the proof of Lemma 1. There, using $\max_{i=0}^{D}(N_i)/(2)$ mixers, we provide a scheduling algorithm, which achieves the lower bound of the completion time and needs zero storage units. So, we only need to prove that it is also the necessary condition for performing all mixing operations with zero storage units.

First, we show that all mixing operations for nonleaf nodes at the same level must be performed in the same time slot if there are zero storage units. Call this the same-time-slot conclusion.

Base Step: There is only one mixing operation for the root node at level 0, so the same-time-slot conclusion is correct obviously.

Induction Step: For some level i, suppose the same-time-slot conclusion holds true for all levels $j < i$. If the conclusion does not hold true for level i, there must be at least two nonleaf nodes A and B in level i, and say A is produced before B. According to the induction hypothesis, A and B are consumed for other nonleaf nodes at level $i-1$ in the same time slot. Obviously, one storage unit is required for A. By contradiction, we get the same-time-slot conclusion.

Since all mixing operations for nonleaf nodes at the same level must be performed in the same time slot, the number of mixers is $\max_{i=0}^{D}(N_i)/(2)$ as desired. ∎

2) Extension to Different Mixing Durations: In Lemma 2, we have computed the lower bound in the case of different mixing durations. However, since different mixing operations in the same level may have different durations, the number of mixers required for a zero-storage-unit schedule can not be derived from the number of mixing operations in the same level. Instead we design Algorithm 7 to compute the number of mixers and schedule for the case of zero storage units below. The basic idea of scheduling in Algorithm 7 is:

- First perform the mixing operations with the biggest length and then proceed upwards.
- Whenever the lower bound minus the lapsed time equals the length of a mixing operation, perform that mixing operation.

Based on the schedule, each time a droplet is produced, a mixer will be released and the number of mixers will decrease by 1; each time a mixing operation starts, a mixer will be taken up and the number of mixers will increase by 1. The largest number of mixers during the schedule gives the number of mixers required for the case with zero storage units.

Algorithm 7: Mixer-Number-for-Zero-Storage-Units

Input: T *// the binary tree*
Output: M, F *// Number of mixers and schedule*
$F = \emptyset$, $M = 0$
// Compute $i.time$*, the length from the root node to* i
for each node i in T **do**
  if $i = T.root$ **then**
    $i.time = 0$
  else
    $i.time = j.time + T_{\text{mix}}^{j}$ *// j is i's parent node*
  end if
end for
for each nonleaf node i **do**
  $s_i = t_{lb} - k.time$, $e_i = t_{lb} - i.time$ *// start, end time*
  // Here k is i's left child node
  // t_{lb} can be computed using Algorithm 1
end for
Sort all s_i, e_i in decreasing order and if $s_m = e_n$, e_n should be before s_m
$n = 0$
while visit all s_i, e_i in order after sorting **do**
  if the value is s_i **then**
    $n = n + 1$
    $F = F \bigcup \{$Schedule the mixing operation at time s_i to achieve node $i\}$
    if $n > M$ **then**
      $M = n$
    end if
  else
    $n = n - 1$
  end if
end while
return M, F

VII. Example

In this section, we apply our scheduling analysis and algorithms to the PCR analysis. Let $t_{\text{transport}}$ be the transportation time from one electrode to its neighboring electrode. The parameter values we use are: $T_{\text{mix}} = 5$ s, $T_{\text{input}} = T_{\text{output}} = 1$ s, $t_{\text{transport}} = 0.01$ s/electrode. (In the experiments of [5] and [8],

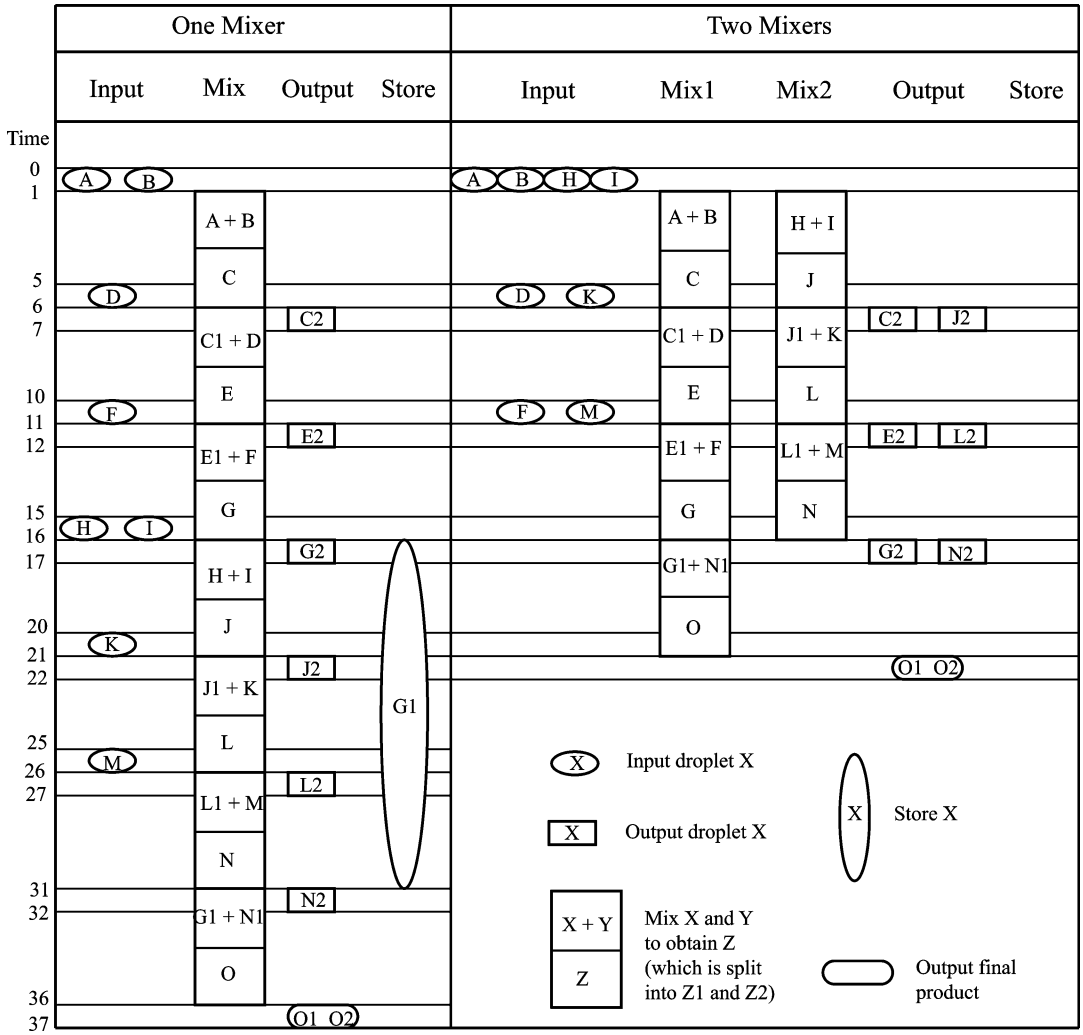


Fig. 14. Scheduling results of performing PCR analysis on DMFS. The left side shows results with one mixer, while the right side shows result with two mixers. Transportation time is ignored here. Accurate completion time considering transportation time is shown in Table I. All times are in seconds.

TABLE I
RESULTS OF PCR ANALYSIS USING ONE AND TWO MIXERS. ALL TIMES ARE IN SECONDS. APPROXIMATE COMPLETION TIME DOES NOT CONSIDER TRANSPORTATION TIME

Num of Mixers	Num of Storage Units	Total Mixing Time	Completion Time	
			Approx.	Exact
1	1	35	37	37.8
2	0	20	22	22.54

$t_{\text{transport}} = 0.006$–$0.013$ s/electrode.) The depth of the binary tree for PCR analysis is $D = 4$.

The results are shown in Table I. Mixing time takes up most of the completion time since pipelining is used to overlap other operations with mixing. Exact completion time (calculated by equation (1)) is almost the same as approximate completion time (calculated by equation (2)) since the transportation speed is very fast. (During one mixing duration, droplets can be transported over $(5)/(0.01) = 500$ electrodes.) The mixing time using two mixers reduces to 57% of that using one mixer due to parallel mixing operations. (The percentage is larger than 50% since one mixer is idle when the last mixing is performed.) By Theorem 1, $T_{\text{lowerbound}} = 20\,\text{s}$, which is the mixing time shown in Table I using two mixers.

We illustrate the scheduling results when one and two mixers are used in Fig. 14. We can see that the pipelining technique is used to overlap different kinds of operations such as *input*, *transport*, *mix*, *store*, and *output*. Comparing the results with one and two mixers, we see that more mixers can exploit the parallelism inside the mixing operations of PCR analysis. By Theorem 1, the lower bound of completion time has been achieved, so more than two mixers cannot decrease the completion time any more.

Other cases of much more complicated biochemical analysis can also be handled in a similar fashion.

VIII. CONCLUSION

In this paper, we presented scheduling algorithms to optimize the completion time of biochemical analyses on a DMFS by exploiting the tree structure of these reactions. Using pipelining techniques, most input and output operations can be overlapped with mixing operations. So, our goal is to complete the mixing operations as soon as possible with the given number of mixers while overlapping other operations with mixing. We introduced a full binary tree structure to model the biochemical analyses.

We calculated the lower bound on the mixing completion time of a biochemical analysis and computed the minimum number of mixers to achieve this lower bound. We also designed a scheduling algorithm of mixing operations for a given number of mixers, and proved it to be optimal in the measure of completion time. Finally, we considered the resource constraint issue for two extreme cases of having just one mixer and having zero storage units. Both the case with identical mixing duration and that of different mixing durations were analyzed and corresponding algorithms were designed. Our results can be applied to various analyses on general-purpose electrowetting-based DMFS platforms, including droplet sensing and analysis operations. In our future work, we will implement our algorithms to control DMFS biochips. We will extend the resource constraint issues and find the relationship between the number of mixers and the number of storage units to guide the chip selection and layout design. Other extensions are to obtain multiple droplets of the final product and to optimize multiple simultaneous reactions in one chip. Combination of layout design, scheduling, and routing algorithms to form integrated systems is also in progress.

ACKNOWLEDGMENT

The authors thank E. Griffith and M. Gupta for early discussions, and R. Linhardt, J. Martin, and J. Dordick for their insights regarding this work.

REFERENCES

[1] P. Dittrich and A. Manz, "Lab-on-a-chip: Microfluidics in drug discovery," *Nature Reviews Drug Discovery*, vol. 5, no. 3, pp. 210–218, Mar. 2006.

[2] B. Zheng, C. Gerdts, and R. F. Ismagilov, "Using nanoliter plugs in microfluidics to facilitate and understand protein crystallization," *Current Opinion in Structural Biology*, vol. 15, pp. 548–555, 2005.

[3] X. X. Chen, H. K. Wu, C. D. Mao, and G. M. Whitesides, "A prototype two-dimensional capillary electrophoresis system fabricated in poly(dimenthylsiloxane)," *Anal. Chem.*, vol. 74, pp. 1772–1778, 2002.

[4] V. Srinivasan, V. K. Pamula, and R. B. Fair, "An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids," *Lab on a Chip*, vol. 5, no. 3, pp. 310–315, Mar. 2004.

[5] S. K. Cho, H. Moon, and C. Kim, "Creating, transporting, cutting, and merging liquid droplets by electrowetting-based actuation for digital microfluidic circuits," *J. Microelectromech. Syst.*, vol. 12, no. 1, pp. 70–80, Feb. 2003.

[6] M. G. Pollack, R. B. Fair, and A. D. Shenderov, "Electrowetting-based actuation of liquid droplets for microfluidic applications," *Appl. Phys. Lett.*, vol. 77, no. 11, pp. 1725–1726, Sep. 2000.

[7] P. Paik, V. K. Pamula, and R. B. Fair, "Rapid droplet mixers for digital microfluidic systems," *Lab on a Chip*, vol. 3, pp. 253–259, Sep. 2003.

[8] R. B. Fair, V. Srinivasan, H. Ren, P. Paik, V. Pamula, and M. G. Pollack, "Electrowetting-based on-chip sample processing for integrated microfluidics," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2003, pp. 779–782.

[9] L. Luo and S. Akella, "Optimal scheduling of biochemical analyses on digital microfluidic systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, San Diego, CA, Oct. 2007, pp. 3151–3157.

[10] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys* pp. 406–471, 1999.

[11] J. Ding, K. Chakrabarty, and R. B. Fair, "Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays," *IEEE Trans. Comput.-Aided Design of Integrated Circuits Syst.*, vol. 25, no. 2, pp. 329–339, Feb. 2006.

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

[13] F. Su and K. Chakrabarty, "Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips," in *Proc. IEEE/ACM Design Autom. Conf.*, 2005, pp. 825–830.

[14] K.-F. Böhringer, "Modeling and controlling parallel tasks in droplet-based microfluidic systems," *IEEE Trans. Comput.-Aided Design of Integr. Circuits Syst.*, vol. 20, no. 12, pp. 1463–1468, Dec. 2001.

[15] E. J. Griffith and S. Akella, "Coordinating multiple droplets in planar array digital microfluidic systems," *Int. J. Robot. Res.*, vol. 24, no. 11, pp. 933–949, Nov. 2005.

[16] E. J. Griffith, S. Akella, and M. K. Goldberg, "Performance characterization of a reconfigurable planar array digital microfluidic system," *IEEE Trans. Comput.-Aided Design of Integr. Circuits Syst.*, vol. 25, no. 2, pp. 340–352, Feb. 2006.

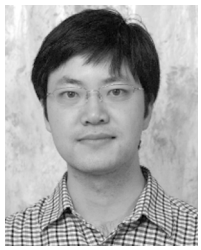
[17] F. Su, W. Hwang, and K. Chakrabarty, "Droplet routing in the synthesis of digital microfluidic biochips," in *Proc. Design, Autom. Test in Europe (DATE) Conf.*, Munich, Germany, Mar. 2006, pp. 323–328.

[18] F. Su and K. Chakrabarty, "Module placement for fault-tolerant microfluidics-based biochips," *ACM Trans. Design Autom. Electron. Syst.* , pp. 682–710, 2006.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York: McGraw-Hill, 2001.

[20] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. San Francisco, CA: Morgan Kaufmann, 2002.

[21] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.



Lingzhi Luo (S'07) received the B.E. degree in automation in 2004, the M.S. degree from the Department of Computer Science and Technology in 2006, both from Tsinghua University, Beijing, China, and the M.S. degree from the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, in 2008. He is currently working towards the Ph.D. degree at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

His current research interests include digital microfluidic systems, scheduling algorithm design, multiagent systems, and machine learning.

Srinivas Akella (S'90–M'00) received the B.Tech. degree in mechanical engineering from the Indian Institute of Technology, Madras, in 1989 and the Ph.D. degree in robotics from the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, in 1996.

He is currently an Associate Professor with the Department of Computer Science, University of North Carolina at Charlotte. He was previously on the faculty of the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY. He was a Beckman Fellow at the Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign prior to that. His research interests are in developing geometric and optimization algorithms for robotics, automation, design, and bioinformatics applications.

Dr. Akella is a recipient of the National Talent Search Scholarship from the Government of India, and of the NSF CAREER award.