DTIP 25-27 April 2012, Cannes, France

Synthesis of Biochemical Applications on Flow-Based Microfluidic Biochips using Constraint Programming

Wajid Hassan Minhass, Paul Pop, Jan Madsen
Technical Univ. of Denmark, DK-2800 Kgs. Lyngby
Phone: +45 4525 3470, Fax: +45 4593 0074, email: whmi@imm.dtu.dk

***Abstract*—Microfluidic biochips are replacing the conventional biochemical analyzers and are able to integrate the necessary functions for biochemical analysis on-chip. In this paper we are interested in flow-based biochips, in which the flow of liquid is manipulated using integrated microvalves. By combining several microvalves, more complex units, such as micropumps, switches, mixers, and multiplexers, can be built. We propose a constraint programming (CP) based approach for the synthesis of biochemical applications on flow-based microfluidic biochips. We use a sequencing graph to model the biochemical application and consider that the biochip architecture is given. We model the architecture using a topology graph. We are interested in synthesizing an implementation, consisting of binding and scheduling of the biochemical operations onto the components of the architecture, such that the resource and dependency constraints are satisfied and the application completion time is minimized. Our CP framework generates optimal implementations and has been evaluated using synthetic as well as real-life case studies.**

***Index Terms*—CAD, Microfluidics, biochips, synthesis, performance**

I. INTRODUCTION

Microfluidics-based biochips have become an actively researched area in recent years. Biochips integrate different biochemical analysis functionalities (e.g., dispensers, filters, mixers) on-chip, miniaturizing the macroscopic chemical and biological processes to a sub-millimetre scale [1]. Microfluidic biochips can readily facilitate clinical diagnostics, enzymatic and proteomic analysis, cancer and stem cell research, and automated drug discovery [1], [2].

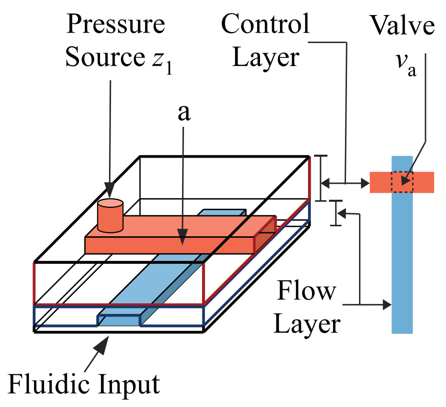
There are several types of microfluidic biochip platforms, each having its own advantages and limitations [3]. In this paper, we focus on the flow-based biochips in which the microfluidic channel circuitry on the chip is equipped with chip-integrated microvalves that are used to manipulate the on-chip fluid flow [1]. By combining several microvalves, more complex units like mixers, micropumps, multiplexers etc. can be built up, with hundreds of units being accommodated on one single chip. This approach is called microfluidic Large Scale Integration (LSI) [1].

Currently, researchers manually map the applications to the valves of the chip using some custom interface (analogous to the exposure of gate-level details in the context of integrated circuits) [4]. The manual process is quite tedious and needs to be repeated every time a change is made either to the chip architecture or the biochemical application. As the chips grow more complex and the need of having multiple and concurrent assays on the chip becomes more significant, these methodologies will become highly inadequate. Therefore, new top-down design methodologies and design tools are needed, in order to provide the same level of CAD support to the biochip designer as the one currently taken for granted in the semiconductor industry.

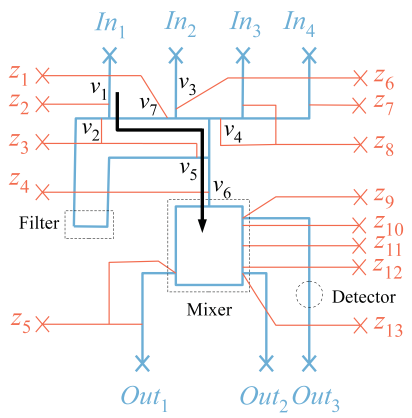
Recently, a top-down design approach for these biochips has been proposed [5]. The main tasks involved in the design process are:

- (i) *Modeling* the biochip architecture, biochip components and the biochemical applications.
- Once the models have been specified, the necessary components for the implementation of the biochemical operations need to be selected from a component library. This is called the (ii) *allocation* step.
- Next the (iii) *placement* of these components and (iv) *channel routing* for interconnecting these components is done.
- This is followed by the decision on (v) *binding* of biochemical operations to the allocated components, and the (vi) *scheduling* step, which determines the time duration for each biochemical operation, while satisfying any resource and dependency constraints.
- Finally, the chip is synthesized according to the constraints on resources, routing channels, area and application completion time. During the chip synthesis, the (vii) *fluid routing* paths from one component to the other component (or from the biochip inputs or to the outputs) also need to be determined.

We have used a topology graph-based model of the biochip architecture and a sequencing graph to model the biochemical application. We consider that the biochip architecture is given, i.e., tasks (ii) to (iv) have been performed. Given a biochip architecture and a biochemical application, we synthesize an implementation, i.e., perform tasks (v) and (vi), such that the application completion time is minimized. In this paper, we do not address task (vii). Our previous research [5] addresses the problem using a List Scheduling (LS)-based heuristic approach. The heuristic based solution uses the sorted and prioritized lists of operations and performs the synthesis (greedy

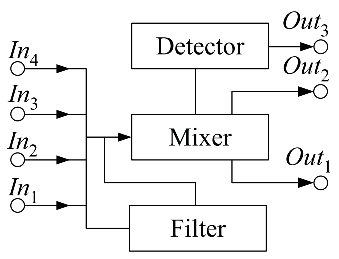


(a) Microfluidic Valve



In_1 In_2 In_3 In_4
z_1 z_2 z_3 z_4 z_5
z_6 z_7 z_8 z_9 z_{10} z_{11} z_{12} z_{13}
v_1 v_2 v_3 v_4 v_5 v_6 v_7
Filter
Mixer
Detector
Out_1 Out_2 Out_3

(b) Biochip: Schematic View



(c) Biochip: Functional View

Fig. 1: Flow-Based Biochip Architecture

binding and scheduling) of ready operations (operations whose predecessor operations have been completed) in a step-wise manner. Every time an operation is completed, the list of ready operations is updated and the newly added operations are synthesized onto the chip. This approach generates good solutions but it neither guarantees solution optimality nor does it provide any measure of the solution quality.

In this paper, we propose a constraint programming (CP) [6] framework which, given a biochemical application and a biochip architecture, determines an optimal solution (in terms of application completion time) for the (v) binding and (vi) scheduling steps. CP makes it possible to specify the resource and timing constraints, and to capture the application binding and scheduling within the same framework. Using the CP formulation, a solver then searches for the optimal solution. CP models are flexible, general and easy to extend. We use synthetic as well as real-life cases to evaluate our approach.

The paper is organized in six sections. We present the biochip architecture details and the biochemical application model in Section II. The targeted problem is discussed and formulated in Section III. The proposed CP-based synthesis approach is presented in Section IV and is evaluated in Section V. Section VI presents our conclusions.

II. System Model

A. Biochip Architecture

Fig. 1b shows the schematic view of a flow-based biochip with four input ports and three output ports, a mixer, a filter and a detector. Fig. 1c shows the functional level view of the same chip. The biochip is manufactured using multilayer soft lithography [1]. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: *flow layer* (depicted in blue) and the *control layer* (depicted in red) [1]. The liquid in the flow layer is manipulated using the control layer.

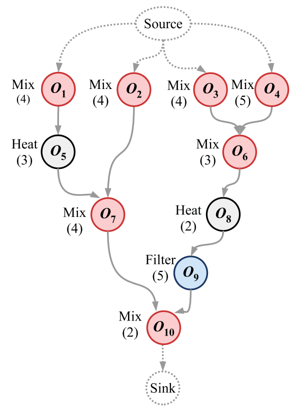
The basic building block of such a biochip is a valve (see Fig. 1a), which is used to manipulate the fluid in the flow layer as the valves restrict/ permit the fluid flow. The control layer (red) is connected to an external air pressure source z_1. The flow layer (blue) is connected to a fluid reservoir through a pump that generates the fluid flow. When the pressure source is not active, the fluid is permitted to flow freely (open valve). When the pressure source is activated, high pressure causes the elastic control layer to pinch the underlying flow layer (point a in Fig. 1a) blocking the fluid flow at point a (closed valve). A biochip can accommodate thousands of valves. By combining these valves, more complex units, such as switches, multiplexers, micropumps, mixers, can be built [3]. We use a topology graph $\mathcal{A}$ in order to capture the biochip architecture. The vertex set $\mathcal{M}$ in the topology graph represents the components present in the biochip, e.g., mixer. More details on the architecture model can be found here [5].

B. Biochemical Application Model

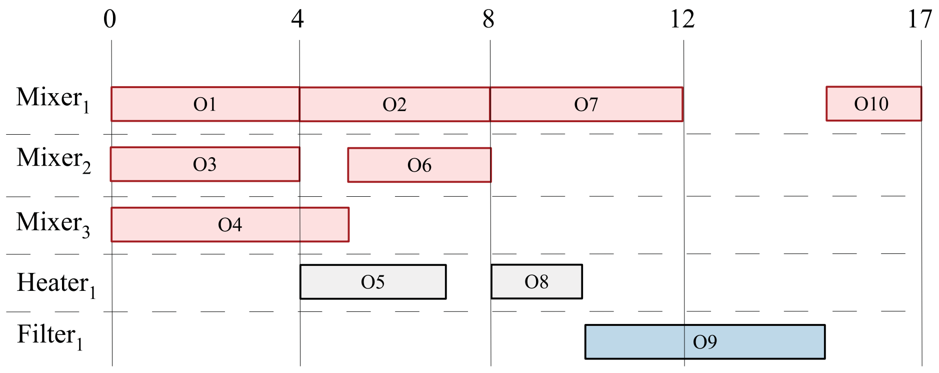
We model a biochemical application using a sequencing graph. The graph $\mathcal{G}(\mathcal{O},\mathcal{E})$ is directed, acyclic and polar (i.e., there is a *source* vertex that has no predecessors and a *sink* vertex that has no successors). Each vertex $O_i \in \mathcal{O}$ represents an operation that can be bound to a component $M_j \in \mathcal{M}$ using a binding function $\mathcal{B}: \mathcal{O} \rightarrow \mathcal{M}$. Each vertex has an associated weight $C_i^{M_j}$, which denotes the execution time required for O_i to be completed on M_j. Fig. 2a shows an example of a biochemical application model which has seven mixing operations (O_1–O_4, O_6, O_7, O_{10}), one filtration operation (O_9) and two heating operations (O_5, O_8). The execution times for the operations are also given in Fig. 2a (the parameter below the operation type).

III. Biochip Synthesis

Implementing a biochemical application onto a biochip architecture requires the following steps: *allocation* of components from a library $\mathcal{L}$, the *placement* of components on a given area, *binding* of operations onto the allocated components, *scheduling* the operations and performing the required fluidic *routing*. The following subsections explain these design



(a) Biochemical Application Model



(b) Schedule

Fig. 2: Illustrative Example

tasks using the biochip architecture given in Fig. 3 and the biochemical application in Fig. 2a.

A. Allocation and Placement

In this paper, we consider that the architecture is given, i.e., the allocation and placement steps have already been performed. The allocated components are captured by the vertex set $\mathcal{M}$ in the architecture model $\mathcal{A}$. For example, for the architecture in Fig. 3, the set $\mathcal{M}$ contains 3 mixers, 2 heaters and 1 filter. The component placement and interconnections are captured by the remaining elements of the topology graph $\mathcal{A}$ modeling the architecture [5].

B. Binding, Scheduling and Routing

Fig. 2b shows the schedule for executing the biochemical application in Fig. 2a on the biochip architecture in Fig. 3. The schedule is represented as a Gantt chart, where, we represent the operations as rectangles, with their lengths corresponding to their execution duration. During the binding step, each vertex O_i ($O_i \in O$, representing a biochemical operation in the application model in Fig. 2a) is bound to an available component M_j, i.e., $\mathcal{B}(O_i) = M_j$. For example, the mixing operation O_1 in the application model in Fig. 2a is bound to the component $Mixer_1$ as shown in Fig. 2b. For routing the fluids, the edges of the application graph $\mathcal{G}$ need to be bound to the channels on the chip. We do not consider fluid routing in this paper.

A scheduling strategy is needed to efficiently execute the biochemical operations on the chip components, while considering the dependency and resource constraints captured by the biochemical application and the biochip architecture models, respectively. Dependency means that a certain operation cannot start execution until all of its preceding operations have been completed. For example, in Fig. 2b, operation O_6 bound to $Mixer_2$ starts after all its predecessors (O_3, O_4) are complete. It starts at $t^{start}_{O_6} = 5$ s and takes 3 s, finishing at time $t^{finish}_{O_6} = 8$ s. The time C_i needed to execute an operation O_i, is given by the application model $\mathcal{G}$.

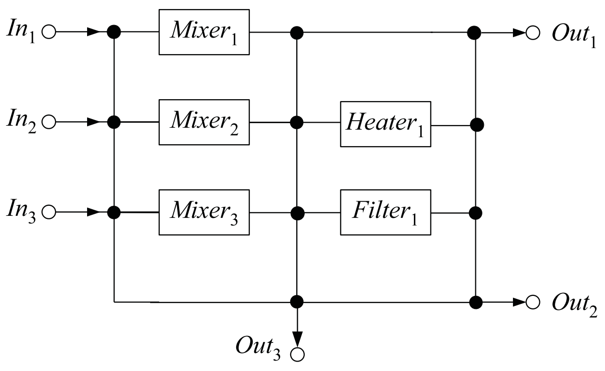


Fig. 3: Biochip Architecture

C. Problem Formulation

The problem addressed in this paper can be formulated as follows: Given (1) a biochemical application modeled as a sequencing graph $\mathcal{G}$ and (2) a biochip architecture modeled as a topology graph $\mathcal{A}$, we are interested in synthesizing an implementation Ψ that optimally minimizes the application completion time while satisfying the dependency and resource constraints. Synthesizing an implementation $\Psi = <\mathcal{B}, \mathcal{X}>$ means deciding on (1) the binding $\mathcal{B}$ of each operation $O_i \in O$ to a component $M_j \in \mathcal{M}$ and (2) the schedule $\mathcal{X}$ of the operations, which contains the start time $t^{start}_{O_i}$ of each operation O_i on its corresponding component.

IV. Constraint Programming-Based Synthesis

The problem can be considered equivalent to the resource constrained scheduling problem with non-uniform weights, which is NP-complete [7]. CP offers very good performance for such problems [6]. Typically, a problem defined in CP has three primary elements: (1) a set of *variables* capturing the system, (2) a set of *finite domains* of the values for these variables and (3) a set of *constraints* imposed on these variables. The *solution* of such a problem is the assignment of values to all variables from their respective domains such that all constraints are satisfied. If an *optimal* solution is desired, then a cost function also needs to be defined in terms of the variables. The solver then finds the optimal solution in terms of the cost function that satisfies all constraints. We have implemented our synthesis approach using the constraint programming environment Gecode [8].

A. Finite Domain Variables

We use the following primary finite domain variables (FDV) to model the binding and scheduling of all biochemical operations:

- $t^{start}_{O_i}$ :: $\{0..\infty\}$ defines the start time of operation O_i. For example, operation O_5 in Fig. 2b has the start time $t^{start}_{O_5}$ = 4 s.
- M_i :: $\{0..\left|\mathcal{M}\right|-1\}$ defines the resource (component) to which the operation O_i is bound where $\left|\mathcal{M}\right|$ is the total number of components on the chip. For example, in Fig. 2b $\left|\mathcal{M}\right|$ = 5. Each resource is assigned a unique numeric identifier (ID) and the range of the identifiers is $0..\left|\mathcal{M}\right|-1$, i.e., 0..4 for the current example. For Fig. 2b, ID of $Mixer_1$ is 0, $Mixer_2$ is 1, $Mixer_3$ is 2, $Heater_1$ is 3 and $Filter_1$ is 4. So for operation O_5, M_5 = 3, which is the ID of $Heater_1$.
- $\delta_{\mathcal{G}}$ :: $\{0..\infty\}$ defines the cost function (application completion time). Application completion time is the end-to-end time taken by the application to complete its execution, i.e., from the start time of the first executed operation to the finish time of the last executed operation. For example in Fig. 2b, the application completion time is $\delta_{\mathcal{G}}$ = 17 s, which is the finish time of the last executed operation O_{10}.

Secondary FDVs are introduced, where needed, in order to implement the constraints.

B. Resource Binding Constraints

An operation can only be bound to a component which is capable of executing it, i.e., a mix operation must be bound to one of the mixers and not to any other component, e.g., heaters. Based on the type of the operations, we constrain the domain of the FDV M_i in order to exclude the forbidden components. Each operation is treated as a tuple (O_i, α_i), where $O_i \in O$ is an operation and $\alpha_i \subseteq \mathcal{M}$ is a set of available components capable of performing the operation. A binding must respect,

$$M_i \in \alpha_i, \quad \forall i \in \{1..|O|\} \tag{1}$$

where $|O|$ defines the total number of operations in the application graph $\mathcal{G}$, e.g., application given in Fig. 2a has 10 operations. For operation O_1 (a mix operation), α_1 = {0, 1, 2} which represents the IDs of the three mixers. For O_1 in Fig. 2b, M_1 = 0 (ID for $Mixer_1$) which is a member of the set α_1, thus satisfying the resource binding constraint.

C. Resource Sharing Constraints

Operations bound to the same component (e.g., O_3 and O_6 are bound to $Mixer_2$ in Fig. 2b) must not overlap in time. We use three disjunctive constraints in order to implement this for all possible combinations of operations.

$$\forall i, \forall(j > i)$$
$$(t^{start}_{O_i} + C_i \le t^{start}_{O_j}) \vee (t^{start}_{O_j} + C_j \le t^{start}_{O_i}) \vee (M_i \ne M_j),$$
$$\forall(i, j) \in \{1..|O|\} \tag{2}$$

where C_i represents the execution time of operation O_i. We consider two operations (O_i, O_j) at a time. At least, one of the three constraints given in the above equation needs to be true in order to ensure that the resources are correctly shared.

The first constraint, $(t^{start}_{O_i} + C_i \le t^{start}_{O_j})$, means that the finish time of operation O_i should be less than or equal to the start time of operation O_j. For example for (O_1, O_2) in Fig. 2b, the finish time for O_1 (4 s) is equal to the start time of O_2, thus the constraint is satisfied. If the first constraint is not satisfied (consider (O_7, O_2) in Fig. 2b, the finish time for O_7 (12 s) is not less than or equal to the start time of O_2 (4 s)), then the second constraint, $(t^{start}_{O_j} + C_j \le t^{start}_{O_i})$, is considered. For ($O_7$, O_2) in Fig. 2b, the finish time for O_2 (8 s) is equal to the start time of O_7 (8 s), thus the constraint is satisfied for this pair of operations. The third constraint, $M_i \ne M_j$, is considered if both the first and second constraint are not satisfied. Consider (O_1, O_3) in Fig. 2b, the finish time for O_1 (4 s) is not less than or equal to the start time of O_3 (0 s) and the finish time for O_3 (4 s) is also not less than or equal to the start time of O_1 (0 s). In this case, the third constraint must be true. Here, M_1 = 0 ($Mixer_1$) and M_3 = 1 ($Mixer_2$), satisfying the resource sharing constraint.

D. Precedence Constraints

During scheduling, an operation must not start executing until its predecessor has completed its execution, e.g., in Fig. 2a, O_6 must finish before O_8 can start. The following constraint is added for each required precedence:

$$t^{start}_{O_i} + C_i \le t^{start}_{O_j} \tag{3}$$

where operation O_i is a predecessor of operation O_j.

E. Cost Function

We are interested in a solution that minimizes the application completion time (cost function $\delta_{\mathcal{G}}$). We constrain the cost function to be greater than or equal to the finish time of the operation O_i, i.e.,

$$t^{start}_{O_i} + C_i \le \delta_{\mathcal{G}}, \quad \forall i \in \{1..|O|\} \tag{4}$$

$$minimize \quad \delta_{\mathcal{G}} \tag{5}$$

Minimization of the $\delta_{\mathcal{G}}$ generates the optimal solution, which satisfies all the imposed constraints. In order to reduce the search space and speed up the result generation, we place an upper and lower bound on $\delta_{\mathcal{G}}$. The upper bound is the sum of execution times of all operations as that is the largest possible application completion time. For the example in Fig. 2a, the upper bound is 36. The lower bound is equal to the duration of

TABLE I: Experimental Results

Application	Allocated components	$\delta_{\mathcal{G}}$	CP execution time
PCR	(2, 0, 0, 0)	16 s	0.2 s
	(3, 0, 0, 0)	16 s	0.2 s
	(4, 0, 0, 0)	12 s	0.2 s
IVD	(2, 0, 0, 2)	25 s	2.6 s
	(5, 0, 0, 5)	18 s	38 min 28 s
	(6, 0, 0, 6)	11 s	1 min 38 s
EA	(2, 1, 1, 0)	19 s	0.3 s
	(3, 1, 1, 0)	17 s	0.4 s
	(4, 2, 1, 0)	17 s	0.5 s

the critical path in the given application graph $\mathcal{G}$. Critical path is defined as the path in the application graph, going from the source nodes to the sink nodes, that has the largest duration. For example in Fig. 2a, the critical path consists of the nodes $\{O_4, O_6, O_8, O_9, O_{10}\}$ and has the duration equal to 17 s. The lower bound value (*lower_bound*) is provided as an input to our CP framework.

$$\delta_{\mathcal{G}} \leq \sum C_i, \quad \forall i \in \{1..|O|\} \tag{6}$$

$$\delta_{\mathcal{G}} \geq lower\_bound \tag{7}$$

V. Experimental Evaluation

We evaluate our proposed approach by synthesizing two real life assays and a synthetic benchmark onto different biochip architectures. The CP framework was implemented in Gecode constraint programming environment [8], running on Lenovo T400s ThinkPad with Core 2 Duo Processors at 2.53 GHz and 4 GB of RAM.

Table I shows our experimental results. Column 1 presents the application and column 2 shows the list of allocated components in the following format (Mixers, Heaters, Filters, Detectors). Column 3 presents the optimal application completion time $\delta_{\mathcal{G}}$ obtained using CP and column 4 presents the time taken by CP to generate the solution.

The first real-life assay is the PCR (polymerase chain reaction) mixing stage that has 7 mixing operations and is used in DNA amplification [9]. We synthesize the assay on three different biochip architectures varying the number of mixers. Each mixing operation is considered to have an execution time of 4 s on the mixing units used in the biochips. As shown in Table I, CP generates optimal solutions using little time.

Multiplexed IVD (in-vitro diagnostics) has a total of 12 operations and is used to test different fluid samples from the human body [9]. As given in Table I, IVD is synthesized onto three different biochip architectures. Each mixing operation is considered to have an execution time of 4 s and the detection operation 7 s. Increasing the number of components reduces $\delta_{\mathcal{G}}$ for IVD from 25 s to 11 s.

The example application (EA) given in Fig. 2a is used as a synthetic benchmark. We vary the number of resources and generate optimal values of $\delta_{\mathcal{G}}$. The CP framework facilitates design space exploration enabling biochip designers to take early design decisions. For example in Table I, the result (row 8 and row 9) shows that increasing the number of mixers from 3 to 4 and the heaters from 1 to 2 does not result in any improvement in $\delta_{\mathcal{G}}$, allowing the designer to use lesser components and reduce chip area. The CP framework can be extended to also consider fluid routing between components.

VI. Conclusions

In this paper we have presented a constraint programming framework for the synthesis of biochemical applications on the flow-based microfluidic biochips. Our approach performs binding and scheduling of the biochemical operations onto the microfluidic components in such a way that all resource and dependency constraints are satisfied. In this work, we generate optimal solutions in terms of minimizing the application completion time. Synthetic as well as real-life examples have been used for evaluating the approach.

VII. Acknowledgments

This work was supported by the Danish Agency for Science, Technology and Innovation, Grant No. 2106-08-0018 “ProCell”.

References

[1] T. Thorsen, S. J. Maerki, and S. R. Quake, “Microfluidic large-scale integration,” *Science*, vol. 298, no. 5593, pp. 580–584, October 2002.

[2] J. M. Perkel, “Microfluidics - bringing new things to life science,” *Science*, November 2008.

[3] D. Mark, S. Haeberle, G. Roth, F. von Stetten, and R. Zengerle, “Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications,” *Chem. Soc. Rev.*, vol. 39, pp. 1153–1182, 2010.

[4] W. B. Thies, “Programmable microfluidics,” *presented at Stanford University*, October 2007.

[5] W. H. Minhass, P. Pop, and J. Madsen, “System-level modeling and synthesis of flow-based microfluidic biochips,” in *Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*, 2011.

[6] K. Kuchcinski, “Constraints-driven scheduling and resource assignment,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 3, pp. 355–383, 2003.

[7] D. Ullman, “Np-complete scheduling problems,” *Journal of Computing System Science*, vol. 10, pp. 384–393, 1975.

[8] C. Schulte, G. Tack, and M. Z. Lagerkvist, “Modeling and programming with gecode.” [Online]. Available: http://www.gecode.org/

[9] F. Su and K. Chakrabarty, “Benchmarks for digital microfluidic biochip design and synthesis,” *Technical Report, Duke University*, 2006.