Simulated Annealing-based Placement for Microfluidic Large Scale Integration (mLSI) Chips

Jeffrey McDaniel  Brendon Parker  Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA, USA 92521

Abstract—Microfluidic large-scale integration (mLSI) chips comprise hundreds or thousands of microvalves integrated into a chemically inert elastomeric substrate. The design of these chips is time-consuming, error-prone, and presently performed by hand. To enhance design automation, a routability-oriented placement algorithm based on simulated annealing is introduced. This paper investigates relevant issues including: (1) grid representation; (2) perturbation operations; (3) objective function; (4) uniform vs. heterogeneous component sizes; (5) spacing rules and their effect on routability; and (6) random vs. directed initial placement. Our results show how the above issues affect both the pre-routing estimate on the routability of the chips, the number of flow channel intersections (each of which requires the insertion of several microvalves), and total channel distance as reported by our router.

Index Terms—Microfluidic large scale integration (mLSI), microfluidics, microvalve, placement, simulated annealing

I. INTRODUCTION

Microfluidic large-scale integration (mLSI) chips automate and miniaturize high throughput biochemical laboratory procedures through a network of microvalves controlled by external pneumatic pressure [8]. Fluid is inserted into the chip from an external pressurized source, and pressure is applied to open and close microvalves in a pre-determined sequence, effectively executing a biochemical program. mLSI chips are used for applications including single-cell imaging, single-cell genomic/proteomic analysis, protein synthesis/characterization, solid-phase liquid chromatography, and many others [3].

At present, mLSI chips are laid by hand using software, e.g., AutoCAD. Automation would be beneficial, as schematic drawing is tedious, non-scalable, and error-prone. mLSI chips may be specified using domain-appropriate hardware design languages in the near future [6], and high-level synthesis tools that convert biochemical protocol specifications into protocol-specific mLSI chips are within reach [7]. It should be possible to leverage techniques from semiconductor VLSI/CAD to automate the design and layout process for mLSI chips.

This paper describes the application of placement techniques for standard cell circuits to mLSI chips. Unlike semiconductor standard cells, which have uniform height and are placed in rows, the components of mLSI chips have arbitrary dimensions and can be placed anywhere on the chip.

Our mLSI placer is based on simulated annealing, and targets a 2-layer soft lithography process [8], with just one layer available for fluid routing (the other layer provides control). Microvalves may be inserted to form switches when fluid channels cross; however, each inserted microvalve may require an additional control input. mLSI chips are I/O limited, so there is a fixed upper bound to the number of fluid channels that may cross, as per microfluidic foundry design rules [13].

A. Abbreviations and Acronyms

An mLSI chip comprises two elastomer layers of polydimethylsiloxane (PDMS), an inert, non-toxic organic polymer widely used for mLSI fabrication, mounted atop a glass slide. The two layers are respectively used for fluidic operations and control. As shown in Fig. 1(a) and (b), cross-points between control and fluid channels form microvalves: applying pressure to the control channel causes the membrane to deflect, closing the fluid channel.

Fig. 1. Cross-sectional (a) and top-down (b) views of an elastomeric mLSI chip. (c) A peristaltic pumping program. (d) a four-step mixing program.
Groups of microvalves and channels form components including switches, pumps, mixers, multiplexers and memories [8]. Components can be defined hierarchically; e.g., three microvalves in sequence form a pump, a sub-component of a mixer. Fig. 1(c) shows a program to operate a 3-valve pump, and Fig. 1(d) shows a 4-step mixing/washing program.

II. PLACEMENT ALGORITHM

We chose simulated annealing for mLSI chip placement due to its long history of success in standard cell VLSI [12] and FPGA placement [16]. Simulated annealing can handle complex constraints and multiple cost functions. We assume that the reader is familiar with simulated annealing.

A. I/O Region Segregation

The placer represents the physical area of the mLSI chip as a two-dimensional grid, from which it can pick coordinates to move components as part of the perturbation operation of simulated annealing. In Fig. 2(a), the grid is partitioned into four region types: input, output, center, and invalid. The placer is constrained to place I/O punches (holes) in the input and output regions respectively, and other components in the center. The invalid regions are isolated in the corner of the chip and no components or I/Os may be placed there; placing components in the chip corners often leads to routing failures, and this possibility is disallowed.

We represent the mLSI chip (components and fluidic connections) as a netlist [6, 9]. We assume that all components occupy one rectangular unit of grid area; Section II.D relaxes this assumption. Non-rectangular components, e.g., the mixer in Fig. 1(d), are represented by a bounding-box. As channel routing follows placement, we assume that all channels start and end in the central points of their terminal components, as shown in Fig 3(a); Fig. 3(b) shows the layout after routing.

\[ \text{Equations:} \quad \text{Let } G = (V, E) \text{ be a graph of the netlist of the chip to be placed. Let } e = (p_i, p_j) \text{ be an edge connecting the center points } p_i \text{ and } p_j \text{ of two components, where point } p_k = (x_k, y_k) \text{ is defined in terms of the } x \text{- and } y \text{- coordinates. Let } N \text{ be the number of intersections in } G. \text{ The intersection point between two non-parallel infinite lines defined by segments } e_1 = (p_i, p_j) \text{ and } e_2 = (p_k, p_l) \text{, respectively, is the point } p^* = (x^*, y^*), \text{ where} \]

\[ x^* = \frac{\sum_{i=1}^{N} x_i y_j - \sum_{j=1}^{N} x_j y_i}{\sum_{i=1}^{N} x_i - \sum_{j=1}^{N} y_j}, \quad \text{and} \quad y^* = \frac{\sum_{i=1}^{N} x_i y_j - \sum_{j=1}^{N} x_j y_i}{\sum_{i=1}^{N} y_i - \sum_{j=1}^{N} y_j} \]

\[ D(e) = |x_1 - x_2| + |y_1 - y_2|. \]
The total edge length $L$ and total squared edge length $S$ are
\[ L = \sum_{e \in E} D(e), \text{ and } S = \sum_{e \in E} D(e)^2. \] (3)

The objective function, $F(G)$, is computed as follows:
\[ F(G) = \alpha N + \beta L + \gamma S, \] (4)
where $\alpha$, $\beta$, and $\gamma$ are user-specified weights, which can be modified to alter the relative importance of each metric.

**Implementation Details:** The time complexity to compute the number of edge intersections is $O(n^2)$; however, recalculating the cost for each edge after each move is unnecessary. We use a two-part cost function to speed up the annealing process. First, each edge maintains a list of the other edges that it intersects. During a perturbation, any change in the set of intersecting edges exclusively involves edges incident on the components that move. $F(G)$ is computed explicitly after the initial placement; subsequently, it is incrementally updated.

Let $E'$ be the subset of edges that are incident on the component(s) that are moved during a perturbation. Let $N'$ denote the number of edge intersections involving edges in $E'$,
\[ L' = \sum_{e \in E'} D(e), \text{ and } S' = \sum_{e \in E'} D(e)^2. \] (5)

After moving all components involved in the perturbation, let $N''$, $L''$ and $S''$ denote the updated values of $N'$, $L'$ and $S'$ that are recomputed. Then the updated metrics are:
\[ N_{\text{new}} = N - N' + N'', \] (6)
\[ L_{\text{new}} = L - L' + L'', \] and
\[ S_{\text{new}} = S - S' + S'', \] (8)
yielding an updated value for $F(G)$, denoted $F_{\text{new}}(G)$. If $F_{\text{new}}(G) < F(G)$, then the perturbation is accepted; if not, it may be accepted probabilistically [12].

**D. Heterogeneous Component Geometries**

Until now, we have assumed that all components have uniform size; the placer preemptively pads smaller components with extra cells to ensure that all components are of the same size, as shown in Fig. 4(a) and (b). This simplified the perturbation process, but yielded poor area usage for mLSI architectures with a large number of small components relative to others. In some cases, the placer could not find legal solutions for small mLSI chips easily placed by hand using exact non-uniform component sizes.

Henceforth, assume that components are rectangular with height and width equal to an integer multiple of the grid pitch. The grid pitch is the greatest common divisor of component widths and heights and the minimum spacing width between components (a foundry design rule). Component dimensions can then be recomputed as integer multiples of the grid pitch.

**Swaps:** Let a block be a square in the grid. The placer randomly selects a component $C$ and a block $b$, which must be outside of the region encompassing $C$. If $b$ is contained within a different component, $D$, then the placer sets $b$ to be the top-left corner of $D$. Next, the placer creates a region $R$ with $b$ as the top-left corner, and dimensions equal to the maximum dimensions of $C$ and $D$. If $R$ contains blocks from multiple grid regions (e.g., input/center) then the swap aborts. If all components with at least one block in $R$ are fully contained in $R$, then the swap is legal; all components within $R$ are swapped with $C$, as shown in Fig. 5. If any component with at least one point in $R$ is not fully contained within $R$, then the swap is aborted, as shown in Fig. 6.

**Shifts:** The placer randomly selects a component $C$ and a direction $d$ (up, down, left, right). The shift tries to move $C$ one block in the direction $d$. A region $R$ is created starting in direction $d$ from the top-left of $C$. If $R$ contains no components other than $C$, the shift is legal (Fig. 7(a)); otherwise it is aborted (Fig. 7(b)).

**E. Component Spacing**

To ensure that components have a valid placement, there must be space between them to accommodate fluid channels for routing. Unlike standard cells, it is not possible to dedicate different layers of metal (PDMS) for logic (components) and wires (fluid routing channels). The minimum spacing requirement to ensure successful fabrication is a design rule provided by the foundry [13]. Spacing beyond the minimum may be necessary to achieve routability.
Let $\Delta$ be the spacing constraint, $C_h/C_w$ be the height/width of the component, and $B_h/B_w$ be the height/width of a block in the grid. Then the number of blocks required for the height ($H$) and width ($W$) of a component, including spacing, is:

$$H = \left\lceil \frac{C_h + \Delta}{B_h} \right\rceil \quad \text{and} \quad W = \left\lceil \frac{C_w + \Delta}{B_w} \right\rceil.$$  \hfill (9)

We found that the spacing rules from Eq. (9) worked fairly well, but led to routing failures when the netlist contained components that connect to a large number of fluid channels.

Let $f_C$ be the number of fluid channels connected to component $C$. To account for $f_C$, the component’s height ($H_f$) and width ($W_f$) are

$$H_f = \left\lceil \frac{C_h + \Delta f_C}{B_h} \right\rceil \quad \text{and} \quad W_f = \left\lceil \frac{C_w + \Delta f_C}{B_w} \right\rceil.$$  \hfill (10)

This provides sufficient spacing to route fluid channels around components with a high degree of fluidic connectivity.

In Fig. 8(a)/(b), two components of equal size have equal padding, regardless of the number of I/O ports. In Fig. 8(c), a component with four I/O ports, instead of two, gets an extra layer of padding to improve routability.

### F. Initial Placement

We consider two heuristics to select an initial placement.

**Random Initial Placement:** Components are processed from largest to smallest to minimize failures due to fragmentation. For each component, the placer determines if a valid position is available; it generates a random location for the component in an appropriate region (I/O or center): if the location is legal, then the component is placed there; if not, another random location is generated. The process repeats until a legal location is found; placement fails if no valid positions are found.

**Directed Initial Placement:** The first component is placed in the top-left position in the region. The algorithm iterates over the list of outgoing edges, and each component is placed nearby in an appropriate region (input, output, center). It maintains variables to represent the starting position of the I/O and center regions in the grid. When a component is selected, the variable for its compatible region selects the next available location. A component’s dimensions determine the number of grid blocks in the horizontal/vertical directions to search. The search proceeds until a valid placement or the end of the region is found; placement fails in the latter case. This process repeats until all components are placed, as shown in Fig. 9.

III. EXPERIMENTAL RESULTS

The placer was implemented as the first step in an mLSI CAD flow under development. We used a set of netlists made publicly available by researchers at the Technical University of Denmark [1]. We placed each netlist using simulated annealing and routed them using a variant of Hadlock’s algorithm for mLSI technology, as suggested by Minhass et al. [10]. We used standard parameter values for simulated annealing. Our initial temperature is 100 degrees, with a 5% cooling rate. We perform 10,000 successful moves at each temperature, and stop when the temperature reaches 1 degree. We report the results of our implementation decisions using the following metrics:

**Success Rate:** We place each circuit 10 times with 10 different random number seeds. The success rate is the number of times Hadlock’s algorithm successfully routes the placed chip.

**Number of Intersections:** We report the number of intersections estimated by the placer, e.g., Fig. 3(a), and the actual number of intersections after routing, e.g., Fig. 3(b). Results are averaged across all (successful) runs. We chose Hadlock’s algorithm for routing to be consistent with prior work [10]; we believe that other algorithms may be better.

**Total Channel Length:** The sum of the lengths of all fluid channels after placement and routing, i.e., term $L$ in Eq. (3).

### A. Objective Function and Random vs. Directed Initial Placement

Our first experiments (Fig. 10(a)/(b)) vary the weights of the objective function parameters using random and directed initial placement. Two trends are observed: (1) random initial placement yields higher success rates than directed initial placement; and (2) equally weighting the number of intersections and total edge length combined with random initial placement generally yielded the best overall results.

The rest of our experiments exclusively use random initial placement and objective function $F(G) = 500N + 500L$. 

---

Fig. 7. A successful shift (a) and an unsuccessful shift (b).

Fig. 8. Components receive equal padding, regardless of the number of I/O ports (a)/(b); extra padding added to a component with four ports to improve routability (c).

Fig. 9. The first four steps of a directed initial placement.
B. Impact of I/O Padding on Components

Fig. 11 reports the impact of scaling the I/O padding around components in accordance with Eq. (10). I/O padding improves the success rate, suggesting that it is ideal for challenging netlists to route. Routing with Hadlock’s algorithm increases the number of intersections over the placer’s estimate. For Synthetic 1-3, which were easier to route, I/O padding increased the number of intersections and marginally increased the total routing channel length; for Synthetic 4-5, which were harder, I/O padding reduced the number of intersections after routing and shortened the total routing channel length.

C. Impact of I/O Region Segregation

Fig. 12 reports the impact of I/O region segregation on the success rate and number of fluid channel intersections. I/O region segregation increases the success rate and reduces the number of intersections (post-routing) for all benchmarks. For Synthetic 5, all routing attempts without segregation failed. This experiment was performed with I/O padding.

D. Summary

To summarize, the best configuration of the simulated annealing placer, based on our experiments, uses: (1) random initial placement; (2) objective function $F(G) = \alpha N + \beta L + \gamma S$, where $N$ is the number of intersections, $L$ is the sum of the edge lengths of all fluid channels, and $S$ is the sum of the squares of the edge lengths of all fluid channels;

The runtime of the simulated annealing placer ranged from 2.3 minutes (Synthetic 1) to 18.6 minutes (Synthetic 5). These runtimes are not absolute, and depend on the temperature schedule, as well as other simulated annealing parameters.

IV. RELATED WORK

One prior paper on mLSI physical design automation claimed to use simulated annealing for placement, but did not provide implementation details [10]. Another described an mLSI placer based on iterative cluster expansion [15], but presented no results. The focus of this latter paper was application mapping, and the placer’s objective was to minimize the length of the longest routed channel to reduce the overhead of fluid transport; it did not consider the cost of microvalves inserted due to intersections introduced by the router. As human intervention required for set-up, teardown, and disposal of mLSI chips in the laboratory, reducing the length of a long fluid channel by a few millimeters is unlikely to have a significant impact on the day-to-day productivity of a biologist who performs experiments using mLSI chips. Meanwhile, limiting the control overhead speaks directly to the feasibility of fabrication, as per microfluidic foundry design rules [13]. By limiting the number of control inputs introduced due to post-routing fluid channel intersections the length of the longest fluid channels could be taken into consideration as a secondary criteria for optimization.
Prior work on mLSI CAD includes application mapping [9, 14], architectural synthesis [10], control synthesis [2, 4, 11], hardware design languages [6, 7], and testing [5]; however, these papers do not address mLSI physical design.

V. CONCLUSION

This paper described a simulated annealing-based placer for mLSI chips. Although we borrowed many ideas from standard cell placement in semiconductor VLSI, the lack of multi-layer metallization and standard cells creates differentiation. We have shown that the choice of objective function, selectively padding components with large connectivity with additional free space, and creating segregated regions for I/O positively impacts the success rate of the router and the number of channel intersections. Future work will study channel routing, as our results showed that Hadlock’s algorithm, as suggested by others [10], was ineffective in practice for mLSI chips.

VI. ACKNOWLEDGMENT

This work was supported in part by National Science Foundation grants 1035603 and 1351115. B. Parker was supported by the California Alliance for Minority Participation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

REFERENCES

[1] Microfluidic Flow-Based Biochips Benchmark Suite (website); URL= https://sites.google.com/site/smmlbiochips/