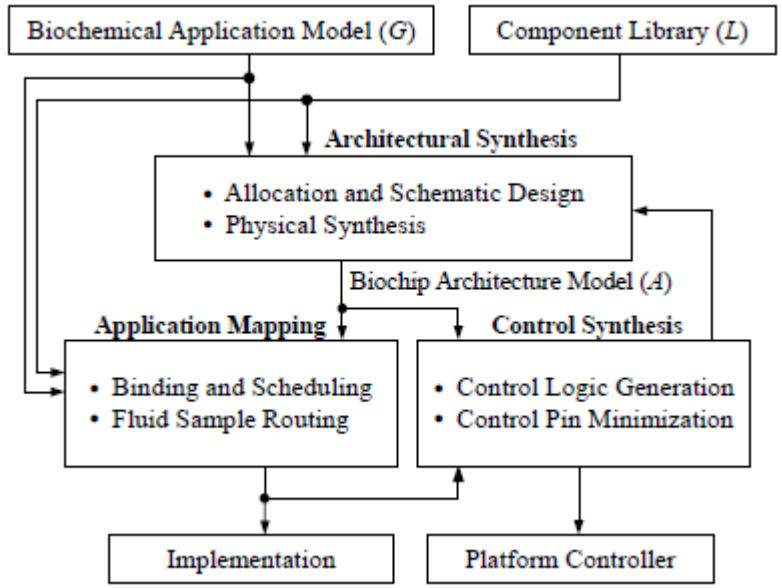
Synthesis of Flow-Based Biochip Architectures from High-Level Protocol Languages

Mathias Kaas-Olsen
B.Sc. Thesis, DTU
August 12th 2015

Introduction

- Flow-based microfluidic biochips.
- Design methodology:
  - Full-custom bottom-up.
  - Top-down by Minhass.
- My focus:
  - Application model synthesis.
  - Architecural synthesis.



Application Model Synthesis

High-Level Language: Aqua

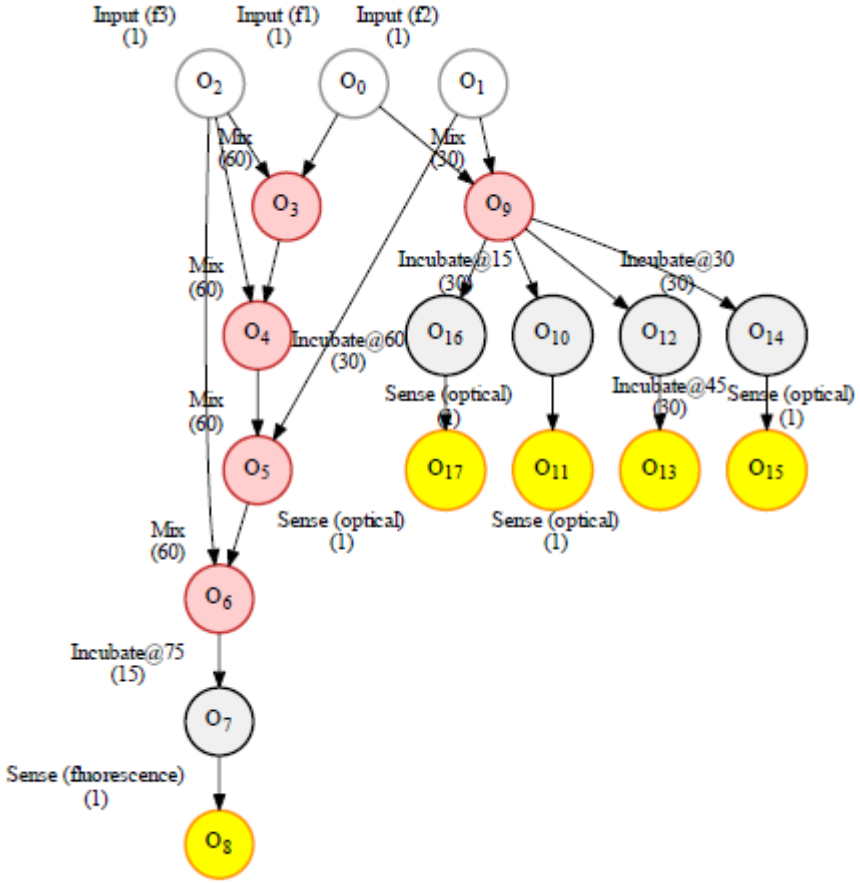
- Declarations:
  - Fluids.
  - Integers.
- Statements:
  - Control-flow (loops).
  - Operations (mix, etc.)

```
ASSAY example START
    FLUID f1;
    FLUID f2;
    FLUID f3;
    FLUID f4;

    VAR v1;
    VAR v2;
    VAR v3[4];

    INPUT f1;
    INPUT f2;
    INPUT f3;

    MIX f1 AND f2 AND f3 IN RATIOS 1 : 3 : 7 FOR 60;
    INCUBATE it AT 75 FOR 15;
    SENSE FLUORESCENCE it INTO v1;

    f4 = MIX f1 AND f2 FOR 30;
    FOR v2 FROM 1 TO 4 START
        INCUBATE f4 AT 15*v2 FOR 30;
        SENSE OPTICAL it INTO v3[v2];
    ENDFOR
END
```

- Language is deterministic at compile-time.

Application Model

- Directed, acyclic graph:
  - Vertices are operations.
  - Edges are dependencies.
- Duration of operations is modelled as the weight of vertices.



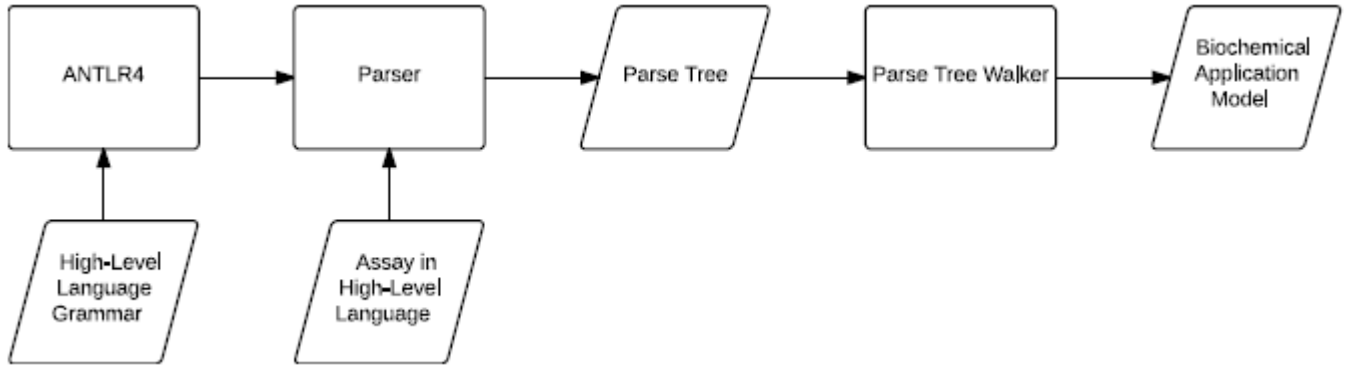
Problem Formulation

- Application Model Synthesis Problem:
  - Given an assay written in Aqua, derive the application graph.

- Mixing Problem:
  - Derive the graph such that it can be executed on architectures with only one-to-one mixers.

Solution Overview

- Define Aqua grammar in EBNF.
- ANTLR generates a parser.
- Parser builds a parse tree from Aqua code.
- Parse tree is traversed to extract operations and dependencies, deriving the application model.

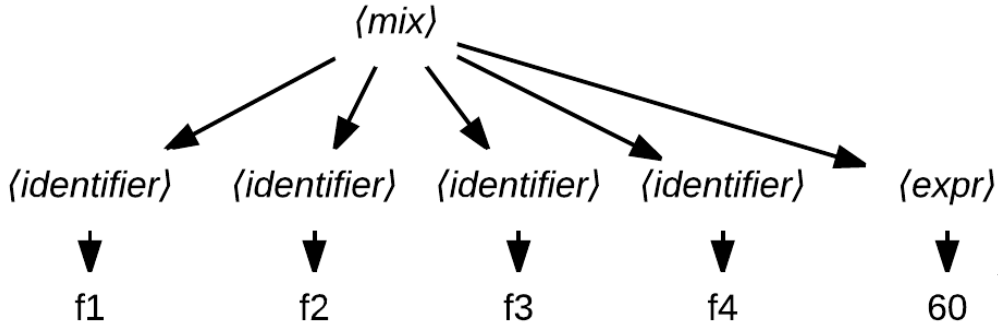


Parse Tree Traversal

- Begin with empty application graph G.
- Recursively visit nodes in the parse tree.
- Tracking variable values:
  - Integer variable value.
  - Fluid variable operation vertex.
- Statements:
  - Control-flow: Visit child statements multiple times.
  - Operation: Add vertices and edges to application graph.

Example: Visit a mix node.

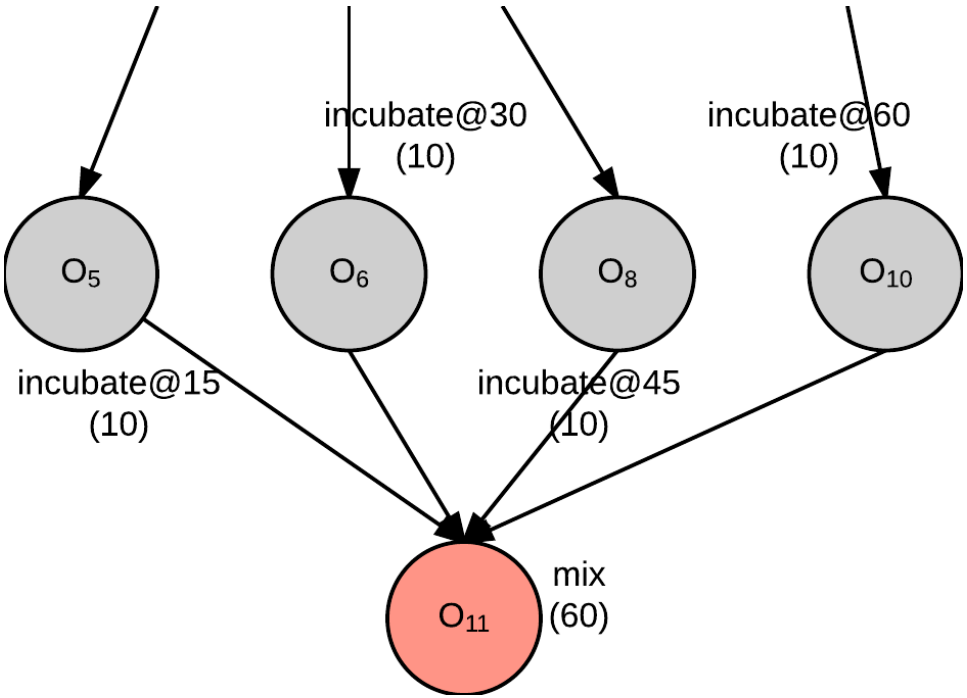
Parse tree fragment:



Variable values:

Fluid Variable	Operation Vertex
f1	O_5
f2	O_6
f3	O_8
f4	O_{10}

Application graph fragment:



Mixing Problem

- Problem: Determine a sequence of 1:1 mixes which achieve a mixture of given ratios.
- Not every set of ratios are reachable using 1:1 mixers. Ratios are reachable if and only if they sum to a power of two.
- Approximate reachable ratios. Approximation can come arbitrarily close to desired ratios.
- Having reachable ratios, use Min-Mix algorithm to determine desired mixing sequence.

Approximation Algorithm

- First determine a target power of two which we will approximate the ratios to.
- Map ratios linearly into target ratio range. Ratios no longer integers.
- Round off ratios to get integers. Ratios are approximations and no longer sum to target.
- Add difference from sum to target by adding or subtracting one from ratios giving least error.
- Absolute error is at most one for every ratio. Relative error compared to desired ratio tends towards zero as the target sum increases towards infinity.

Example

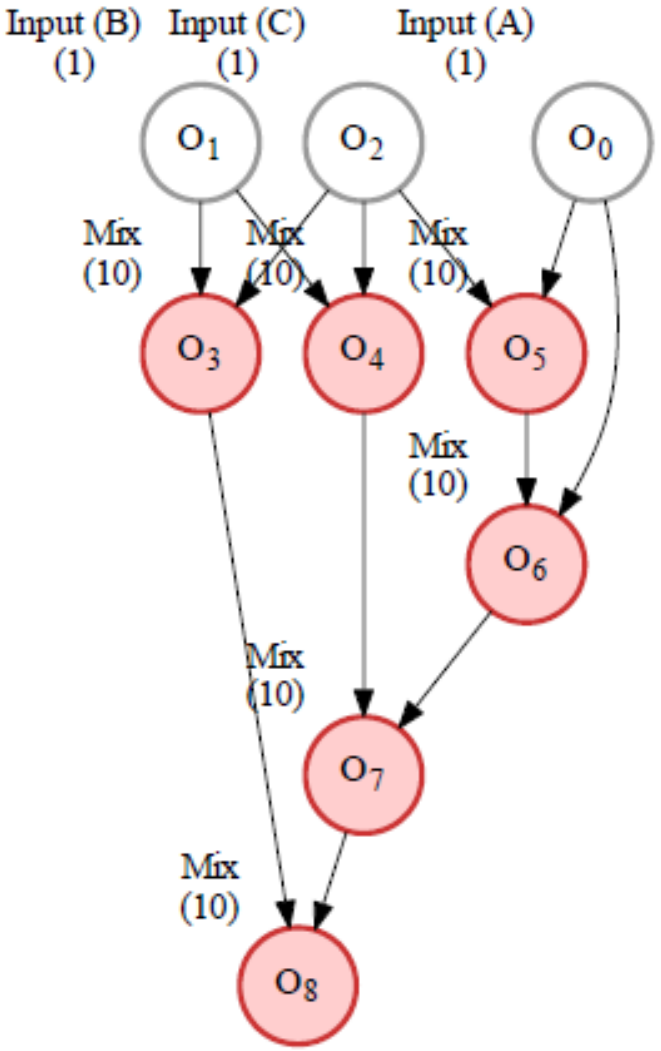
- Given unreachable ratios (2, 5, 6), sum 13.
- Choose target power of two: 16.
- Linearly map ratios into new range: (2.46, 6.15, 7.38).
- Round off ratios: (2, 6, 7), sum 15.
- Rounded sum off by 1 from target.
- Add 1 to ratio which gives least error: (3, 6, 7).

Min-Mix Algorithm

- Construct mixing tree by observing that each mixing operation halves contribution of a fluid to the final mixture.
- Use this observation to determine at what depths of mixing tree to put a leaf for each input fluid.
- Connects leaves to form internal nodes, representing the sequence of mixes required to obtain desired mixture.
- Runtime and mixing tree size: O(n log2 R) where n is number of fluids and R is sum of ratios.

Example

- Mixture of (A,B,C) in ratios (3, 6, 7).
- A: 3/16 = 1/16 + 2/16,
  3 is 011 in binary.
- B: 6/16 = 2/16 + 4/16,
  6 is 110 in binary.
- C: 7/16 = 1/16 + 2/16 + 4/16,
  7 is 111 in binary.
- Ordinal positions of ones in binary representation determines the depths at which leaves are placed.



Arcitectural Synthesis

Component Library and Constraints

- Component library:
  - Characterise components.
  - Name, functions, size, etc.

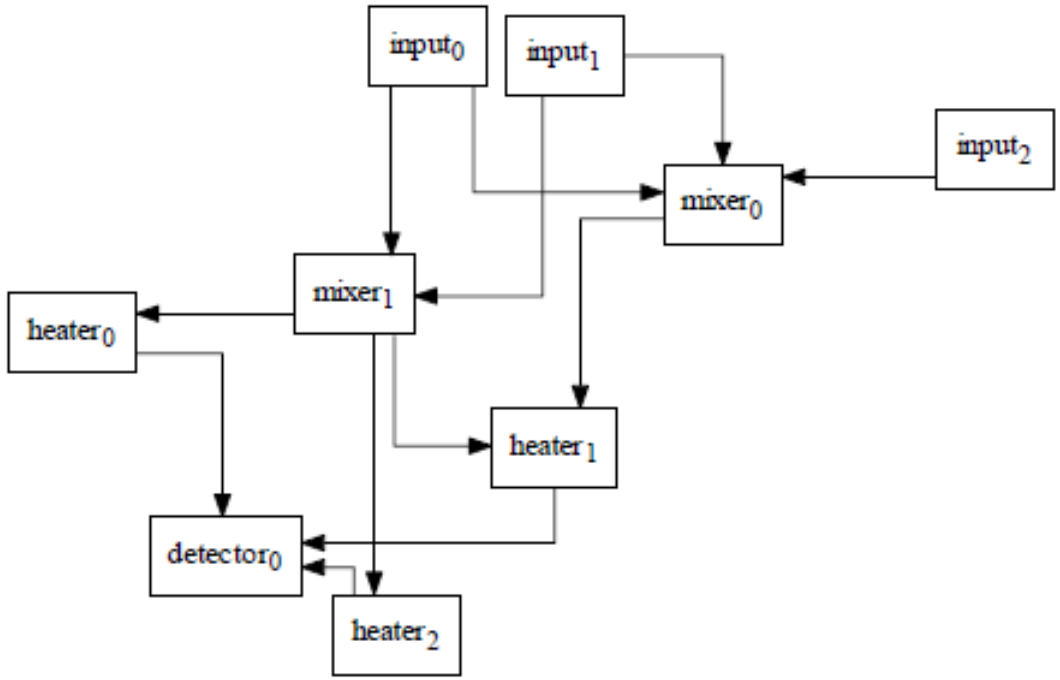
Name	Type	Size	Valves
Input	-	(5, 5)	1
Output	-	(5, 5)	1
Filter	filter	(120, 30)	2
Heater	heat	(40, 15)	2
Mixer	mix	(30, 30)	9
Detector	detect	(20, 20)	2
Separator	separate	(70, 20)	2
Metering	-	(30, 15)	6
Multiplexer	-	(30, 10)	2
Storage	-	(90, 30)	28
Switch	-	(1, 1)	3
SwitchI	-	(1, 1)	2
SwitchT	-	(1, 1)	3
SwicthX	-	(1, 1)	4

- Constraints:
  - A set of restrictions for number of allocated components of each type.

Name	#components
Filter	0
Heater	3
Mixer	3
Detector	1
Separator	0

Architecture Model

- Netlist - directed graph:
  - Vertices are components.
  - Edges are connections.



Problem Formulation

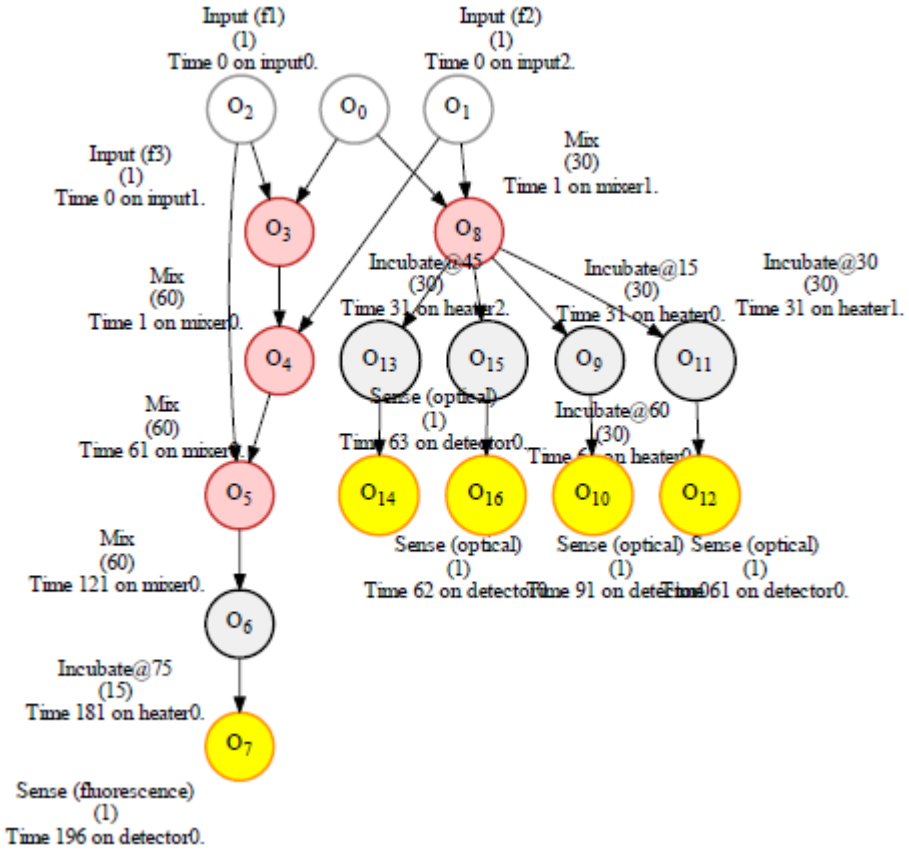
- Architectural Synthesis Problem:
  - Allocation and Schematic Design: Given an application graph, a component library and a set of resource constraints, derive a biochip architecture to efficiently perform the application.

Allocation

- Resource-constrained list-based scheduling algorithm; determine preliminary binding and scheduling:
  - Prioritize operations on urgency criteria: length of longest path to sink node.
  - Repeatedly determine list of ready operations.
  - Greedily schedule operations on already allocated components.
  - Allocate more components if an operations is ready and we're not violating the constraints.

Example

- Scheduled and bound application graph:

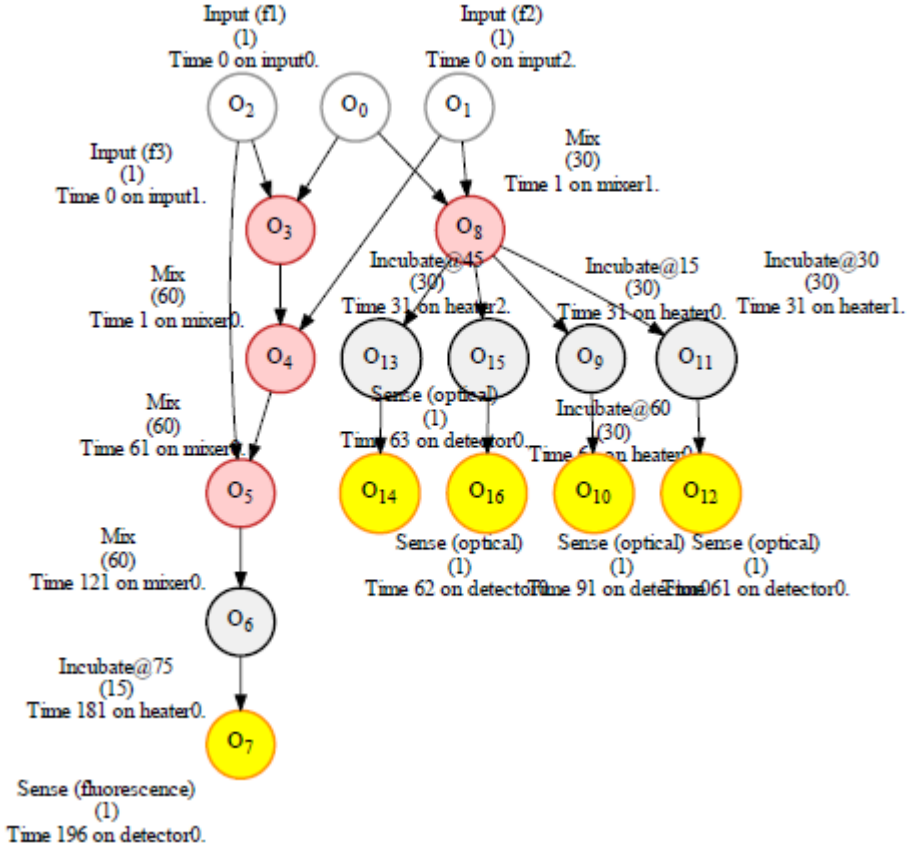


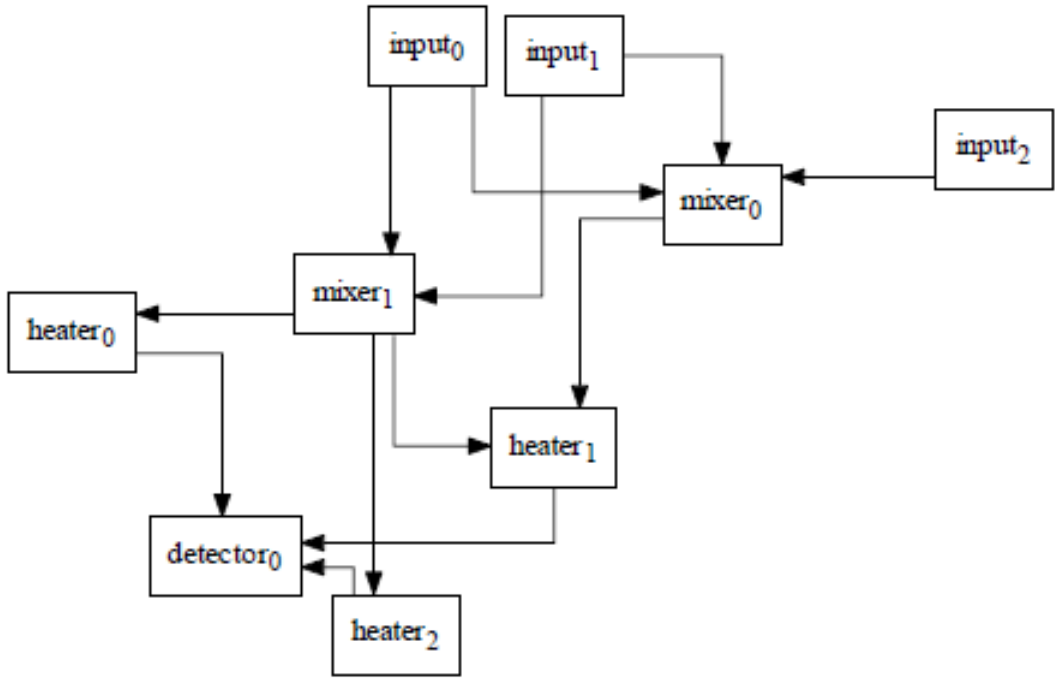
Schematic Design

- Use application graph along with preliminary binding to derive netlist.
- For each allocated component, add a vertex to the netlist.
- For each dependency between two operations in the application graph, add an edge between the two components to which the operations were bound.

Example

- Netlist derived from bound application graph:

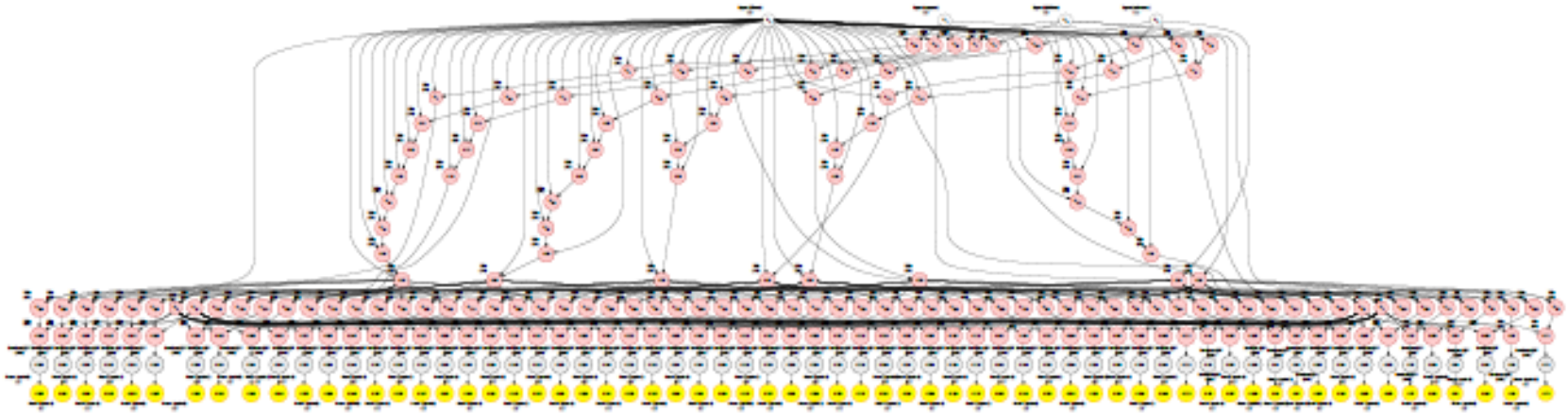




Results, Conclusion and Future Work

Results

Assay	Lines	Total time (seconds)	Operations
simple_mix	10	0.10	3
complex_mix	11	0.10	9
example	24	0.17	17
enzyme_test	57	0.39	326
glucose_test	30	0.19	17



Results

-x	Approximation	Errors	Operations
0	[3, 6, 7]	[21.9, -2.5, -5.2]%	9
1	[5, 12, 15]	[1.6, -2.5, 1.6]%	10
2	[10, 25, 29]	[1.6, 1.6, -1.8]%	11
3	[20, 49, 59]	[1.6, -0.5, -0.1]%	12
4	[39, 99, 118]	[-1.0, 0.5, -0.1]%	15
5	[79, 197, 236]	[0.3, 0.0, -0.1]%	16

Conclusion

- We have a tool which synthesises the application model from an assay in Aqua.
  - The application graph can be tailored to execute on architectures with 1:1 mixers for any number of fluids and mixing ratios.
  - Approximation of ratios can be arbitrarily accurate.
- We have a tool which synthesises an application-specific allocation and schematic design of a biochip architecture. The efficiency is hard to determine.

Future Work

- Extend High-Level Language to add more operations.
- Extend application graph to capture intrinsic details of operations.
- Mixing algorithms for other ratios than 1:1.
- Fluid Volume Management during Application Model Synthesis?