Statistical Optimisations of Asynchronous Dynamic Dataflow

Paulo Garcia, Heriot-Watt University
Robert Stewart, Heriot-Watt University

In this paper, we pose the following question: can we infer any understanding about a dataflow network that allows us to optimise (i.e., schedule) it, when dynamic, non-deterministic behaviour is prevalent?

CCS Concepts: •**Hardware** → **Reconfigurable logic and FPGAs; High-level and register-transfer level synthesis; Hardware description languages and compilation; Logic synthesis;** •**Software and its engineering** → **General programming languages; Architecture description languages;**

ACM Reference Format:
Paulo Garcia, Robert Stewart. Statistical Optimisations of Asynchronous Dynamic Dataflow *ACM Trans. Architec. Code Optim.* 9, 4, Article 39 (March 2016), 8 pages.
DOI: 0000001.0000001

1. INTRODUCTION

The dataflow model(s) of computation is a paradigm that can be used to implement, represent and reason about myriad processes. Dataflow models processes as concurrent computational blocks (actors) communicating through data (tokens) sent across point-to-point channels. This paradigm is applicable to hardware pipelines, parallel software, distributed systems, etc.; and has been used to implement a broad range of solutions, using one of the many dataflow sub-models, depending on the nature of the application.

The different models of dataflow differ in two key areas: synchronicity and token throughput. *Synchronicity* refers to the timing semantics of different actors: are their actions scheduled according to a global time frame, which can be predicted statically? Or are execution semantics unpredictably variable throughout runtime? *Token throughput* refers to token consumption and production rates: do actors consume/produce a fixed number of tokens per action (or cyclic sequences of tokens), which can be used to reason about global throughput? Or is the throughput unpredictable as well?

There is an inversely proportional relation between predictability and applicability. The simplest models, which exhibit highly predictable execution semantics, can be used to design and represent equally simple systems: e.g., synchronous state-less hardware. The most complex model -dynamic asynchronous- can be used to design and represent virtually any system at any scale (e.g., distributed computing), at the cost of predictability: i.e., it is more complex to reason about its execution semantics, namely timing.

Timing analysis, however, is of paramount importance to *optimisation functions*: manual or automated methods that consume a dataflow network and produce and optimised version. Re-designing or refactoring a dataflow system to optimise a particular metric (e.g., performance, power consumption) requires trustworthy assumptions

This work is supported by

DOI: 0000001.0000001

about the behaviour of that system. State of the art timing analysis methodologies, however, are based on formal semantics that rely on static token throughput and synchronisation.

In this paper, we investigate the timing analysis of dynamic asynchronous dataflow and its application to optimisation functions. We abandon the notion of precise timing knowledge, and instead build statistical timing models that can guide optimisation functions. Specifically, this paper offers the following contributions:

— We identify and discuss the limitations of state of the art timing analysis techniques for asynchronous dynamic dataflow, and how these limitations prevent optimisations across a broad range of dataflow implementations.
— We present a methodology for statistical timing analysis based on Probablity Density Functions (PDFs), which can be applied to asynchronous dynamic dataflow.
— We present a methodology that applies PDF analysis to dataflow optimisations, namely scheduling strategies for software implementations and clock gating strategies for hardware implementations.

The remainder of this paper is organized as follow: in Section 3, we revise the different types of dataflow models and the state of the art in dataflow timing analysis, and highlight the current limitations of applying such techniques to asynchronous dynamic dataflow. In Section 4, we present our methodology for statistical timing analysis of asynchronous dynamic dataflow. In Section 6, we describe how our timing analysis can be applied to scheduling optimisations of dataflow software implementations, and in Section 7 we describe how it can be applied to clock gating optimisations of dataflow hardware implementations. In Section 8, we evaluate our approaches on a suite of dataflow applications, implemented on CPU and FPGA. Finally, we present our conclusions and identify future work in Section 9.

2. INTRODUCTION2

Can we infer any understanding about a dataflow network that allows us to optimise (i.e., schedule), when dynamic, non-deterministic behaviour is prevalent?

3. BACKGROUND AND RELATED WORK

In order to have this paper self-contained, we review three dataflow models: static synchronous, cyclo-static synchronous, and dynamic asynchronous (the interested reader may refer to [Mirza et al. 2014] and [Bouakaz et al. 2017] for a more detailed discussion of other models such as cyclo-static asynchronous). Although other model variants exist, we focus our exposé on these three types, which suffice to illustrate the key aspects of interest in this paper.

— different types of dataflow, focusing on dynamic asynchronous
— related work in timing estimation, token critical path analysis
— examples of why it doesn't work for dynamic asynchronous

4. PDF CRITICAL PATH ANALYSIS

As previously mentioned, the timing analysis of synchronous dataflow is predicated on predictable token consumption/production rates at discrete, globally synchronised points in time. This discrete scheduling may be a global clock transition in hardware implementations or a scheduling time slice in software implementations. On asynchronous dataflow, this discrete abstraction must be abandoned: absolute continuous time must instead be adopted. This is true even in Globally Asynchronous Locally Synchronous (GALS) hardware implementations: even though each actor operates under its own (discrete) clock, clock frequencies may be completely unrelated in frequency

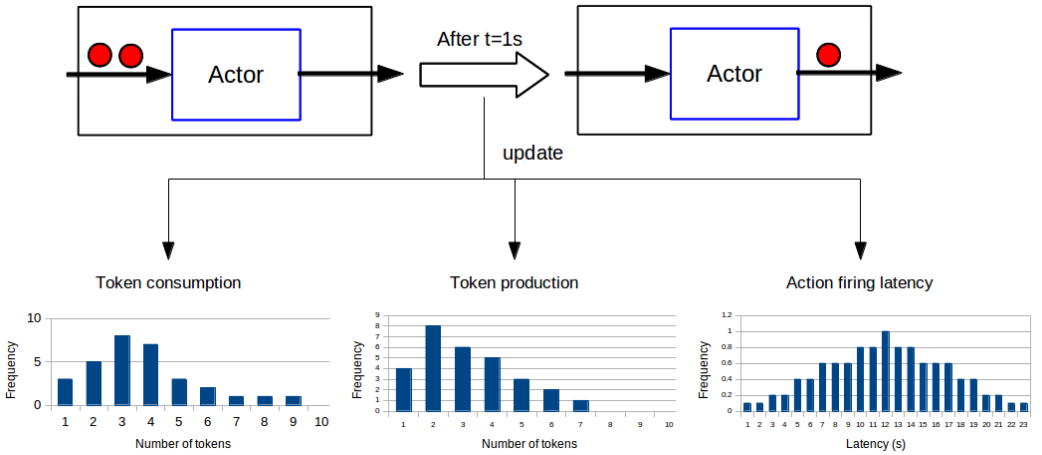

Fig. 1. Profile update after action firing: token consumption is updated by incrementing bin 2, token production is updated by incrementing bin 1, and latency is is updated by incrementing bin 1. Histogram shapes are purely illustrative.

and phase. The goal of PDF analysis is to obtain a token throughput timing profile in absolute time, under unpredictable token consumption/production rates. This is done through network profiling, collecting runtime statistics about each actor's behaviour.

4.1. Single Input Single Output Actors

For the simplest possible dynamic asynchronous actor, with one input and one output port, we monitor how many tokens are consumed and produced per action firing, and how long it takes between successive action firings, assuming sufficient input tokens are always available. It should be noted that we assume these two metrics to be independent: the dynamic nature of actors doen't allow for any assumptions of correlation between token consumption and latency. Fig. 1 depicts an example of such profiling, for different token throughput and action latencies, and examples of histograms for each metric.

A simplistic analysis of actor behaviour would be the average time between token consumption. We can define that one token is consumed every t_T seconds, where:

$$t_T = \frac{\sum_0^{n_T} H(T)}{\sum_0^{n_L} H(L)} \tag{1}$$

In order to model actors' behaviour in absolute continuous time,

— each actor latency (avg of actions) as a PDF
— effect of pipeline analysis of PDF
— effect of feedback loops analysis of PDFs

5. MODEL

A dataflow network can be modeled as the following matrix equation:

$$\mathbf{P} = \mathbf{UC} \tag{2}$$

where **C** represents token consumption ports, **P** represents token production ports and **U** represents the utilisation of connections between producers and consumers. It can be re-written as:

$$\mathbf{A_p P} = \mathbf{U'A_c C} \tag{3}$$

where **U'** represents the new utilisation of connections between producers and consumers, and $\mathbf{A_c}$ and $\mathbf{A_p}$ map consumer/producer ports to actors according to actor execution ratio (by default, identity matrix, i.e., all actors run during the entire time), respectively, such that:

$$\begin{pmatrix} a_{p1} & 0 & 0 & \cdots & 0 \\ 0 & a_{p2} & 0 & \cdots & 0 \\ 0 & 0 & a_{p3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{pm} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_m \end{pmatrix} = \begin{pmatrix} u_1 & 0 & 0 & \cdots & 0 \\ 0 & u_2 & 0 & \cdots & 0 \\ 0 & 0 & u_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_m \end{pmatrix} \begin{pmatrix} a_{c1} & 0 & 0 & \cdots & 0 \\ 0 & a_{c2} & 0 & \cdots & 0 \\ 0 & 0 & a_{c3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{cm} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{pmatrix} \tag{4}$$

This can be re-written as the linear system:

$$\begin{cases} p_1 a_{p1} = u_1 a_{c1} c_1 \\ p_2 a_{p2} = u_2 a_{c2} c_2 \\ \vdots \\ p_m a_{pm} = u_m a_{cm} c_m \end{cases} \tag{5}$$

Hence, the problem of scheduling a dynamic dataflow network, can be re-written as: *choosing a **U'** diagonal matrix, such that the linear system in (9) is solvable, and such that the non-zero elements of **U'** are as close as possible to 1.* The last element in the definition can be formally described as:

$$min \sum_{n=1}^{m} \sqrt{(u_{n,n} - 1)^2} \tag{6}$$

Two heuristics are important to simplify this analysis:

$$\forall x : 0 < a_x \leq 1 \cap \forall n : 0 < u_{n,n} \leq 1 \tag{7}$$

since actor execution ratio must be between 0 and 1, and queue utilization should have an upper limit of 1 to prevent full queues and corresponding deadlocks.

5.1. Classification Matrices

Having modeled a general dataflow network and specified the requirements for the existence of a solution (that the linear system in (9) is solvable) and formalised the problem within linear algebra, we can specify classification matrices (pairs of $\mathbf{A_c}$ and $\mathbf{A_p}$) which characterise particular dataflow network configurations, and analyse the problem for each classification.

5.1.1. Single port feed forward. Single port feed forward networks are the simplest type of dataflow networks. In this class, each actor has only one input and one output port: hence, the full network has one input and one output, resulting in the classification matrices:

$$\mathbf{A_p} = \begin{pmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_m \end{pmatrix}, \mathbf{A_c} = \begin{pmatrix} a_2 & 0 & \cdots & 0 \\ 0 & a_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{8}$$

These matrices are characterised by having $\mathbf{A_{p1,1}} = 1$, $\mathbf{A_{cm,m}} = 1$ and no repeated actor mapping elements. Additionally, it is possible to name actors in such a way that actor mapping elements are ordered in both matrices (this is not possible in networks with feedback). The linear system solution for single port feed forward networks results in:

$$\begin{cases} p_1 a_1 = u_1 c_1 a_2 \\ p_2 a_2 = u_2 c_2 a_3 \\ \vdots \\ p_m a_m = u_m c_m \end{cases} \tag{9}$$

or, in augmented matrix form:

$$\left(\begin{array}{cccc|c} p_1 & -u_1 c_1 & 0 & \dots & 0 \\ 0 & p_2 & -u_2 c_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & p_m & u_m c_m \end{array}\right) \tag{10}$$

Since both the coefficient and augmented matrices have rank m (the number of free variables), the system has a unique solution for any values of **U'**: hence, it is always possible to set all queue utilisations to 1 and determine actor execution ratios.

5.1.2. Divergent feed forward. In divergent feed forward networks, each actor has only one input port, but may have several output ports, resulting in networks that diverge into two or more independent paths: hence, the full network has one input and several outputs, resulting in the classification matrices:

$$\mathbf{A_p} = \begin{pmatrix} a_{\mathbf{1}} & 0 & \cdots & 0 \\ 0 & a_{\mathbf{1}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_m \end{pmatrix}, \mathbf{A_c} = \begin{pmatrix} a_2 & 0 & \cdots & 0 \\ 0 & \ddots & \cdots & 0 \\ \vdots & \vdots & \mathbf{1} & \vdots \\ 0 & 0 & \cdots & \mathbf{1} \end{pmatrix} \tag{11}$$

These matrices are characterised by having repeated elements in $\mathbf{A_p}$, and the same number of 1s in $\mathbf{A_c}$ as the number of repetitions in $\mathbf{A_p}$.

There is no assurance that these types of networks have a solution: however, it is still possible to infer knowledge about the network and optimise it, even when the system is not solvable. In order to do that, we divide the network into k paths, where k is the number of repetitions in $\mathbf{A_p}$; each path corresponds to a single port feed forward network. Then, actor execution ratios are calculated by solving the system. This results in three possible cases:

(1) Results for actor execution ratios are consistent across all k paths. In this case, there is a single solution that optimises the network.

(2) Results for actor execution ratios are not consistent across all k paths, but all results are ≤ 1. In this case, it is possible to optimise the path with the highest ratio for the corresponding divergent actor, and update lower ratio paths accordingly (e.g., if path 1 results in $a_1 = 0.8$ and $a_2 = 1$, and path 2 results in $a_1 = 0.4$ and $a_3 = 0.2$, then the final ratios are $a_1 = 0.8$, $a_2 = 1$ and $a_3 = 0.2 \times \frac{0.8}{0.4} = 0.4$).
(3) Results for actor execution ratios are not consistent across all k paths, applying the previous update technique yields at least one result > 1. This means that an over-utilised queue was present in the network. Since this results in blocked actors (correspondingly affecting output throughput in some path), it is more efficient to reduce actor execution ratio to 1 across the over-utilised path, and modify other ratios accordingly. This still results in a lower throughput than desired, but at a smaller energy cost.

5.1.3. Convergent feed forward. In convergent feed forward networks, each actor has only one output port, but may have several input ports, resulting in networks that converge from two or more independent networks: hence, the full network has several inputs and one or more outputs, resulting in the classification matrices:

$$\mathbf{A_p} = \begin{pmatrix} \mathbf{1} & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{1} & 0 & \cdots & 0 \\ 0 & 0 & a_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_m \end{pmatrix}, \mathbf{A_c} = \begin{pmatrix} a_{\mathbf{1}} & 0 & 0 & \cdots & 0 \\ 0 & a_{\mathbf{1}} & 0 & \cdots & 0 \\ 0 & 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & 1 & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{12}$$

These matrices are characterised by having repeated elements in $\mathbf{A_c}$, and the multiple 1s in $\mathbf{A_p}$.

5.1.4. Convergent/divergent feed forward. Convergent/divergent feed forward networks exhibit the characteristics of both previous types: hence, the classification matrices also exhibit both previous characteristics:

$$\mathbf{A_p} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{1} & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_{\mathbf{1}} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & a_{\mathbf{1}} & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_m \end{pmatrix}, \mathbf{A_c} = \begin{pmatrix} a_1 & 0 & 0 & \cdots & 0 \\ 0 & a_{\mathbf{2}} & 0 & \cdots & 0 \\ 0 & 0 & a_{\mathbf{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \mathbf{1} & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{1} \end{pmatrix} \tag{13}$$

5.1.5. Feedback.

5.2. Example

Consider the example dataflow network depicted in Fig. 2: this simple network would result in the following matrix system $\mathbf{P} = \mathbf{UC}$:

$$\begin{pmatrix} 5 \\ 8 \\ 4 \\ 10 \end{pmatrix} = \begin{pmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 10 \\ 4 \\ 2 \\ 10 \end{pmatrix} \tag{14}$$

where the values in the $\mathbf{U}$ matrix tell us the utilization of connected queues: e.g., queue a is under-utilized (ratio is smaller than 1), meaning that more tokens are being consumed than produced onto the queue, whilst all other queues are over-utilized (ratios are greater than 1), meaning that more tokens are produced than consumed from

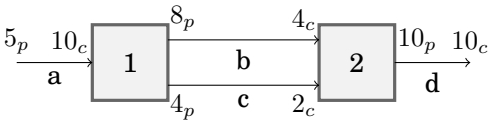

Fig. 2. Do not forget! Make it explicit enough that readers can figure out what you are doing.

the queue. Matrices $\mathbf{A_c}$ and $\mathbf{A_p}$ can now be introduced (derived from the connections of consumer/producer ports to actors or network inputs/outputs), where, by replacing **U** with **U'**, which is the desired topology matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & a_1 & 0 & 0 \\ 0 & 0 & a_1 & 0 \\ 0 & 0 & 0 & a_2 \end{pmatrix} \begin{pmatrix} 5 \\ 8 \\ 4 \\ 10 \end{pmatrix} = \begin{pmatrix} u1 & 0 & 0 & 0 \\ 0 & u2 & 0 & 0 \\ 0 & 0 & u3 & 0 \\ 0 & 0 & 0 & u4 \end{pmatrix} \begin{pmatrix} a1 & 0 & 0 & 0 \\ 0 & a2 & 0 & 0 \\ 0 & 0 & a2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 \\ 4 \\ 2 \\ 10 \end{pmatrix} \tag{15}$$

Setting **U'** as the identity matrix, solving for a_1 and a_2 results in the linear system:

$$\begin{cases} 5 = 10a_1 \\ 8a_1 = 4a_2 \\ 4a_1 = 2a_2 \\ 10a_2 = 10 \end{cases} = \begin{cases} 10a_1 = 5 \\ 8a_1 - 4a_2 = 0 \\ 4a_1 - 2a_2 = 0 \\ 10a_2 = 10 \end{cases} \tag{16}$$

which can be re-written as the augmented matrix:

$$\left(\begin{array}{cc|c} 10 & 0 & 5 \\ 8 & -4 & 0 \\ 4 & -2 & 0 \\ 0 & 10 & 10 \end{array}\right) \tag{17}$$

Since the rank of the coefficient matrix and the rank of the augmented matrix are both the same, and identical the number of unknowns, this system has unique solution:$a_1 = 1/2$ and $a2 = 1$: i.e., actor 1 should run for 50% of the time.

In this example, there is a solution to the system when **U'** is the identity matrix; however, this is seldom true.

6. SCHEDULING STRATEGIES

— Assuming round robin scheduling
— Counters to determine where to move to

7. GATING STRATEGIES

— Timers to determine when to gate

8. EXPERIMENTS

9. CONCLUSIONS

ACKNOWLEDGMENTS

We acknowledge the support of the Engineering and Physical Research Council, grant references EP/K009931/1 (Programmable embedded platforms for remote and compute intensive image processing applications) and EP/J015180/1 (Sensor Signal Processing).

REFERENCES

Adnan Bouakaz, Pascal Fradet, and Alain Girault. 2017. A Survey of Parametric Dataflow Models of Computation. *ACM Trans. Des. Autom. Electron. Syst.* 22, 2, Article 38 (Jan. 2017), 25 pages. DOI:http://dx.doi.org/10.1145/2999539

U. M. Mirza, M. A. Arslan, G. Cedersjo, S. M. Sulaman, and J. W. Janneck. 2014. Mapping and scheduling of dataflow graphs: A systematic map. In *2014 48th Asilomar Conference on Signals, Systems and Computers*. 1843–1847. DOI:http://dx.doi.org/10.1109/ACSSC.2014.7094787

Received February 2016; revised March 2016; accepted June 2016