Technische Universiteit
Eindhoven
University of Technology

Performance Analysis of Weakly-Consistent Scenario-Aware Dataflow Graphs

Marc Geilen, Joachim Falk, Christian Haubelt, Twan Basten, Bart Theelen and Sander Stuijk

ES Reports
ISSN 1574-9517

ESR-2011-03
12 December 2011

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Performance Analysis of Weakly-Consistent Scenario-Aware Dataflow Graphs

Marc Geilen[1], Joachim Falk[2], Christian Haubelt[3],
Twan Basten[4,1], Bart Theelen[4] and Sander Stuijk[1]

[1]Dept. Electrical Engineering, Eindhoven University of Technology
[2]Dept. Computer Science, University of Erlangen-Nuremberg
[3]Dept. Computer Science and Electrical Engineering,University of Rostock
[4]Embedded Systems Institute, Eindhoven, The Netherlands

Email: {m.c.w.geilen,s.stuijk, a.a.basten}@tue.nl, falk@cs.fau.de,
christian.haubelt@uni-rostock.de, b.d.theelen@esi.nl

December 12, 2011

Abstract

The timed dataflow model of computation is a useful performance analysis tool for Electronic System Level Design automation and embedded software synthesis. It is used to model systems, including platform mapping and resource scheduling, of components communicating and synchronizing in regular patterns. Its determinism gives it strong analysability properties and makes it less subject to state-space explosion problems. Because of its monotonic temporal behaviour it can provide hard real-time guarantees on throughput and latency. It is expressive enough to cover a fairly large class of applications and platforms. The trend however, in both embedded applications and their platforms is to become more dynamic, reaching the limits of what the model can express and analyse with tight performance guarantees. Scenario-aware dataflow (SADF) is an extension that allows more dynamism to be expressed, introducing a controlled amount of non-determinism into the model to represent different scenarios of behaviour. The combination of a relatively infrequent switching between scenarios and still deterministic dataflow behaviour within scenarios stretches the expressiveness of the model while keeping sufficient analysability. In this report we investigate so-called weakly consistent graphs in which the scenario changes are not tightly coupled with periods of repetitive behaviour of the static dataflow behaviour in scenarios as in previous methods. We define their semantics in terms of $(\max, +)$ algebra and we introduce a method to analyse throughput using a generalisation of $(\max, +)$-automata.

keywords: performance analysis, synchronous dataflow, $(\max, +)$-algebra

1 Introduction

To develop concurrent embedded software applications and the platforms on which they execute, it is important to be able to efficiently assess whether or not performance requirements will be met. The possible parallel tasks and resource arbitrations create dependencies between tasks that manifest themselves as synchronisations between tasks and time delays to capture processing or execution times, data dependencies and synchronization constraints. Such synchronisations and delays are captured well by performance models that build upon the $(\max, +)$ semiring [1], such as Network Calculus [5], Real-Time Calculus [29], timed Petri-nets [8,24], max-plus automata [13], and the timed dataflow models. In this work we use timed dataflow models to study the performance of stream-based applications mapped on a platform under allocated resources.

An important feature of timed dataflow models for performance analysis is their determinacy. Dependencies between tasks cannot change because of variations in timing or scheduling. There exist different classes of dataflow models. In the most restricted ones, such as timed Synchronous Dataflow (SDF) [21, 26], dependencies must also be independent of input data, while some more dynamic models (for instance Dynamic Dataflow [7]) do allow such variation with different input data. A consequence of the fact that the structure of dependencies is fixed, is their monotonicity. Whenever a task requires less time to execute than the modelled worst-case execution time, this cannot lead to a situation where any other task completes its work later. (This would for instance be possible for tasks being scheduled under a non-preemptive scheduling policy.) The great benefit is that while in reality execution timing may be non-deterministic, it can be conservatively, accurately modelled by a completely deterministic execution. This avoids or reduces state-space explosion problems during performance analysis.

The growing challenge is that the static structures of data dependencies and regular execution times with limited variation are becoming more and more exceptional as both applications and platforms are becoming more dynamic. Applications are becoming more dynamic, for instance, because of complex data reduction schemes, which introduce strong data-content dependencies. Video codecs will have very low data rates for static scenes, while data rates may increase a lot for moving action scenes. Hand held devices need to support a wide range and diversity of communication protocols. More and more is handled in software by software-defined radio implementations. Novel cognitive radio protocols have strong adaptivity to environmental conditions. For MP3 compression, different parts of the audio, called frames, may be encoded using different methods. These methods cannot be accurately captured in a single, static dataflow model. Besides the application, also the platforms are becoming more dynamic. They need to dynamically handle various use-case scenarios of applications and use dynamic QoS management to match available resources with applications.

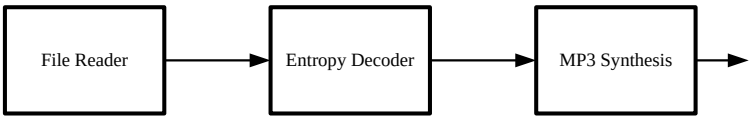
In order to deal with the increasing amounts of dynamic behaviour in applications and platforms, there is a growing need for performance models that can deal with more dynamic behaviour and can still provide tight performance guarantees. Most importantly, they should remain at the same high level of abstraction and they should not fall victim to state-space explosion problems more than necessary. For this reason, the so-called Scenario-Aware Dataflow (SADF) timed dataflow model [28] tries to maintain as much as possible of the determinacy of dataflow behaviour, while introducing the possibility for non-deterministic variations in the

form of *scenarios*. In MP3 decoding for instance, there are five individual coding schemes for audio frames. Each of these schemes can be represented accurately by a static dataflow graph, while the types of frames may occur non-deterministically in arbitrary orders. The SADF model and analysis techniques exploit the determinacy in behaviour within a single scenario, while allowing for non-deterministic selection of the scenarios that occur. A crucial aspect is the concurrency among different scenarios. Concurrent implementations of streaming applications are often pipelined. For the MP3 decoder this means that different frames in different scenarios may simultaneously be decoded. Yet, it turns out that for the analysis, scenario behaviour can be separately and sequentially handled, despite their overlap in time.

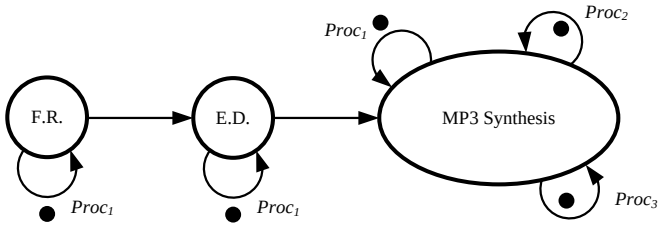
The SADF model, introduced in [28], models how scenarios of behaviour can be detected in an input stream being processed and how the application responds differently to different scenarios. In this way more accurate performance bounds can be computed or estimated. Initial analysis methods were based on the construction of a labelled transition system based on the operational semantics of the model. Because of a limited separation of non-deterministic scenario behaviour and the determinate dataflow computations, the methods could suffer from state-space explosion problems. Analysis methods for the Synchronous Dataflow model, based on spectral analysis techniques in the linear algebra on the $(\max, +)$ semi-ring have been introduced to the analysis of a particular subset of SADF models that are called *strongly consistent*, which means that every individual scenario behaviour corresponds to a complete iteration of an SDF graph. For the analysis of the combination of non-deterministic sequences of scenarios which are modelled as SDF behaviour, the theory of $(\max, +)$-automata [13] has been used [15].

The contribution of this report is to generalise the performance analysis approach of [15] to the case of [28], where scenarios may occur at a finer granularity than complete SDF iterations. This class is called *weakly consistent* SADFs, as opposed to the *strongly consistent* case, in which every scenario corresponds to a full iteration and hence is consistent in the sense that on every dependency edge, the number of produced and consumed data elements during that scenario are equal. For weakly consistent graphs, this is not necessarily the case, although in the long run, they need still be consistent (explained more precisely in Section 4), hence the name 'weakly consistent' graphs. In particular, we introduce methods to determine the worst-case throughput of a weakly consistent SADF and a compact state-space from which latency type of properties can be determined.

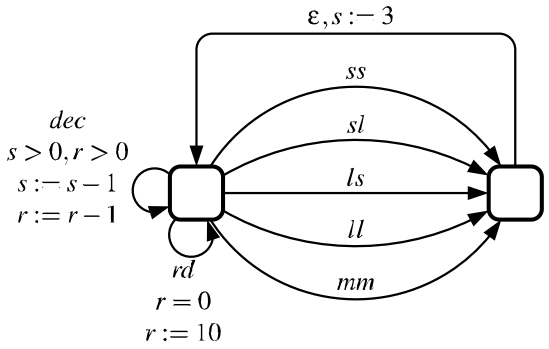
Example An example of the type of system that we are addressing is shown in Figure 1, which depicts an MP3 decoder. Figure 1(a) shows its structure. The decoder gets its input from some data source, e.g., a file, in an input buffer that is being refilled when it becomes empty. The input data is decompressed by an entropy decoder based on an Huffman code. Every now and then an audio frame is completed and can be processed by the synthesis filter banks. Figure 1(b) shows a dataflow model of the decoder. Note that the bigger ellipse in the figure, labelled 'MP3 Synthesis' is in fact a large dataflow graph, which depends on the audio frame type and consists of up to 20 separate actors taking hundreds of actor firings to complete the synthesis of the frame. The three tokens shown on the edges sticking out of the synthesis graph represent the dependencies carried over from one frame to the next. In this example, we have mapped the decoder onto three processors. The three tokens $Proc_i$ represent these resource dependencies. Each processor individually, but independently from each other, needs to complete a frame before starting the next frame. Note that in Figure



(a) block diagram of MP3 decoder



(b) dataflow graph of MP3 decoder



(c) FSM specifying scenario sequences

Figure 1: An MP3 decoder

1(b) there are three tokens labelled $Proc_1$, these in fact represent the same token modeling the processor 1 dependency, but in different scenarios.

Finally, the finite state automaton (on infinite words) in Figure 1(c) is our specification of the possible orders in which scenarios can occur. We have used counters (r and s) to provide a more compact representation of the automaton, but it can easily be unfolded to a regular FSM. Edges are labeled with guards on the counters and assignments to counters, but, most importantly, with the scenario that is executed when the edge is taken. *dec* denotes execution of the entropy decoder, *rd* of the file reader and *ss*, *sl*, *ls*, *ll* and *mm* represent decoding of any of the five different types of audio frame. The file reader needs to be run once every ten firings of the decoder and the decoder may non-deterministically produce a complete frame, but needs to complete a frame at least once every five executions.

Interleaving of non-deterministic choices of the FSM (which, unfolded, has 65 states) with the execution of this dataflow graph (792 firings in the largest scenario, *ss*, as well as pipelining of multiple frames) can easily lead to state-space explosion. Moreover, in this example there are two independent, unsynchronized, sources of non-deterministic behaviour. On the one hand, the entropy decoder occasionally produces a frame to be synthesised, but how often depends on the compression achieved for the particular piece of music. On the other hand,

the decompressed frames occur in different types which are decoded differently. Making this distinction leads to tighter estimates of the performance compared to synchronous dataflow models without scenarios. The results of this report make it easier to model the individual sources of scenarios independently and to model them more accurately. Forcing this behaviour into a synchronous model may require abstractions that leads to less precise performance estimates.

2 Related Work

Within the broad scope of performance analysis and schedulability analysis, we focus on a particular class of systems with repetitive behaviour, processing on streams of data leading to strong dependencies between the individual tasks. The $(\max, +)$ semi-ring and its linear algebra [1] are very suitable to express the behaviour and semantics of such models and it forms the basis of many popular performance analysis models, such as Network Calculus [5], Real-Time Calculus [29], timed Petri-nets [8, 24], and the family of timed dataflow models of which SADF is a member.

Dataflow models of computation come in a wide variety of different versions that range from very static through more dynamic and (partially analysable) models, to very dynamic, but also very hard to analyse models. The static models include for instance (homogeneous, cyclo-static) synchronous dataflow [4, 22, 26] and Computation Graphs [20]. The following models incorporate some dynamism, but still provide a good level of analysability. Heterochronous Dataflow (HDF) [16] introduces dynamism by combining a finite state automaton with synchronous dataflow graphs in the individual states. The model is restricted to executing complete iterations per state transition of the automaton and the model does not have a timed version for performance analysis. Parameterised Synchronous Dataflow (PSDF) [2] considers a static structure of a dataflow graph, where one or more of the port rates are given by parameters rather than constants. It is possible to find parameterized schedules for such graphs and find appropriate buffer sizes, but the possibilities for expressing dynamism are limited. Apart from a similar parameterised model, Wiggers et al. also introduce a var iable rate dataflow model (VRDF) [33], where communication rates of actors may vary arbitrarily and are not necessarily constant over a complete iteration. Analysis methods for this model are restricted to (conservative) buffer sizing under throughput constraints. More dynamic variations on dataflow models have been defined, but they introduce serious difficulties in the analysis. Their buffer sizing and throughput analysis problems are undecidable. Dynamic dataflow (DDF) and Boolean Dataflow (BDF) [6] are dataflow models where the firing rules can be data dependent. Kahn Process Networks [18, 19] are also a form of dynamic dataflow model, but not based on actors with firing rules. Its dynamism and data-dependent behaviour make the relevant analysis problems undecidable.

Network Calculus was introduced for the analysis of network processing on streams of network traffic. It abstracts concrete streams into worst-case bounds on amounts of traffic observed in any interval of a particular duration. Real-Time Calculus is a specialization of the Network Calculus approach to schedulability analysis of real-time embedded systems. In particular towards modeling of arbitration of shared resources and resource composition. The abstraction into the time-interval domain makes it harder to model dependencies, in

particular circular dependencies, although extensions have been made to make handling such dependencies feasible or more accurate [30].

Petri-nets [25], with its many variants is also a model of computation that can express deterministic dataflow behaviour, as well as non-deterministic behaviour. Timed Marked Graphs [8] are in fact a class of Petri-nets equivalent to timed synchronous dataflow graphs. Determinism and consistency can be expressed as network invariants. We are not aware of any Petri-net analysis techniques that combine large aggregations of deterministic dataflow behaviour with only the essential non-deterministic choices.

An appropriate semantic domain for timed synchronous dataflow behaviour is $(\max, +)$-linear algebra [1]. We also exploit it in this report. Spectral analysis in this linear algebra is intimately related to throughput and latency analysis. $(\max, +)$-automata [13] combine $(\max, +)$-linear behaviour with non-deterministic choice. We use this combination to model non-determinism introduced by scenarios. The Heaps of Pieces model [14,32] is a specialisation of $(\max, +)$-automata, used in literature to study the behaviour of discrete event systems and in particular Safe Timed Petri/nets (see for instance [14]). It is important to observe the difference between the $(\max, +)$-automaton model and the Heaps of Pieces model, namely that Pieces cannot accurately capture a larger collection of dataflow actor firings as a single Piece, since Pieces have fixed relative starting times (the 'lower contour') and fixed completion times (the 'upper contour'). As such, a Piece is 'rigid', while an iteration is more 'flexible' as it consists of a collection of independent actor firings, i.e., an aggregated stack of Pieces; the resulting upper contour may depend on the starting lower contour. By modelling only individual firings, the model is too fine-grained to efficiently represent complex graphs.

We will show how our analysis problem is ultimately mapped on a Maximum Cycle Ratio (MCR) problem on a directed multigraph, derived from a $(\max, +)$-automaton. For the MCR analysis we have implemented the algorithm of Young et al. [34]. A generalisation of cycle mean or cycle ratio analysis is provided by spectral analysis of $(\max, +)$-linear systems [9,11]. Spectral analysis gives not only the cycle mean, (eigenvalue), but also an eigenvector, which provides information about the relative firing times of actors, or latency. A good overview and comparison of cycle mean, cycle ratio and spectral analysis methods can be found in [10].

In this report we use synchronous dataflow graphs in the individual scenarios. However, in contrast to most work dealing with synchronous dataflow graphs we do not consider only complete iterations [15, 16], but allow partial repetition vectors. A special case of grouping firings results from clustering of SDF actors [3, 12, 23]. The result can be a quasi-statically scheduled system, in which the clustered actors can be modelled as scenarios of a weakly-consistent SADF. Hence, the proposed analysis can also be applied to such systems. Another work exploring this aspect is [31], which considers a modular implementation of SDF, where firings of an SDF iteration are grouped together. These may be individually scheduled depending on the presence of input data. The work in this report might be used to add a performance dimension to this kind of code synthesis.

3 Preliminaries

We give brief introductions to SDF, the extension of dataflow with scenarios, the timing-semantics of SDF formulated in $(\max, +)$-algebra, and $(\max, +)$-automata as an analysis tool to the minimal extent required for this report.

3.1 Synchronous Dataflow Graphs

Synchronous Dataflow Graphs are directed (multi-)graphs in which the nodes represent *actors*, entities that model computations or other events that take time, such as a computation task on a processor or resource arbitration delays. Actors perform their events or actions repeatedly. A single action is called a *firing* of the actor. The directed edges, connecting actors, are called *channels* and represent dependencies between actor firings. Dependencies in the model may have different origins in reality. They may be data dependencies, but they may also represent resource dependencies, for instance when an activity requires a resource that first needs to be released by another activity. Concrete dependencies are incarnated by *tokens* (sometimes also called *delays*) that are being communicated across the channels. In SDF, actors producing or consuming tokens can do so with constant *rates*. Each firing may consume or produce multiple tokens, but the number needs to remain constant across firings. Channels may initially already contain some tokens, which are called *initial tokens*.

Because of the constant rates with which tokens are communicated, actor firings occur in repetitive patterns called *iterations*. An iteration defines a (smallest, positive) number of firings for each actor which is such that the net effect on the number of tokens on channels remains unchanged. From this invariant, it is clear that this pattern or iteration can be repeated infinitely to obtain a *streaming* execution of the dataflow graph.

3.2 $(\max, +)$ semantics of SDF

Semantics of SDF graphs comes in two flavours. Some focus on functional behaviour of actors and graphs. Others focus on the performance of SDF graphs to predict their throughput or latency. In this report, we investigate the second kind, timed SDF [26]. SDF graphs can be translated into equivalent event graphs [1, 26], although this may involve a considerable increase in the size of the graph. From this it does follow however that the timing behaviour of SDF graphs follows similar patterns as event graphs. In particular, their behaviour is deterministic and eventually becomes periodic. This behaviour can be captured efficiently by means of $(\max, +)$- algebra [1], a linear algebra based on the operations of maximum and addition.

An interesting feature of timed SDF graphs is that although the semantics assumes fixed, deterministic execution times and therefore has a deterministic behaviour, it can faithfully capture systems in which the execution times are non-deterministic, yet bounded from above, by deterministic execution times. Throughput results of the deterministic SDF graph provide guaranteed lower bounds on the actual throughput of the system [1, 26].

The timing is encoded by *dater functions* which assign to tokens alive in the graph at a given state, a time stamp of their first occurrence. We illustrate this with an example in Section 5.

We now briefly introduce some notation related to $(\max,+)$-algebra (see [1] for background on $(\max,+)$-algebra). $(\max,+)$-algebra defines the operations of the maximum of numbers and addition over the set $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$, the real numbers extended with $-\infty$. For readability we use the standard notation for the max and addition operations instead of the $\oplus$ and $\otimes$ notation mostly used in $(\max,+)$ literature. Moreover, for scalars x and y, $x \cdot y$ (with shorthand xy) denotes ordinary multiplication, not the $(\max,+)$ $\otimes$ operator. The max and $+$ operators are defined as usual with the additional convention that $-\infty$ is the zero-element of addition: $-\infty + x = x + -\infty = -\infty$ and the unit element of max, $\max(-\infty, x) = \max(x, -\infty) = x$. $(\max,+)$ is a *linear algebra*: $x + \max(y,z) = \max(x+y, x+z)$. The algebra is additionally extended to a linear algebra of matrices and vectors in the usual way. For a matrix $\mathbf{M}$ and vector $\boldsymbol{x}$, we use $\mathbf{M}\boldsymbol{x}$ to denote the $(\max,+)$ matrix multiplication. $||\boldsymbol{a}||$ denotes a vector norm, defined as: $||\boldsymbol{a}|| = \max_i a_i$, i.e., the maximum element. For a vector $\boldsymbol{a}$ with $||\boldsymbol{a}|| > -\infty$, we use $\boldsymbol{a}^{norm}$ to denote $\boldsymbol{a} - ||\boldsymbol{a}||$, the normalised vector $\boldsymbol{a}$, so that $||\boldsymbol{a}^{norm}|| = 0$. An inner product is defined as follows: $\boldsymbol{a}^T\boldsymbol{b} := \max_i(a_i + b_i)$. If matrix $\mathbf{M} = [\boldsymbol{m}_j]$ (i.e., has column vectors $\boldsymbol{m}_j$), then $\mathbf{M}\boldsymbol{x} := \max_j(\boldsymbol{m}_j + \boldsymbol{x})$ and $\mathbf{M}^T\boldsymbol{x} := [\boldsymbol{m}_j^T\boldsymbol{x}]$. It is easy to verify that also matrix multiplication is linear: $\mathbf{M}(\max(\boldsymbol{x}, \boldsymbol{y})) = \max(\mathbf{M}\boldsymbol{x}, \mathbf{M}\boldsymbol{y})$ and $\mathbf{M}(c + \boldsymbol{x}) = c + \mathbf{M}\boldsymbol{x}$.

3.3 Scenario-Aware Dataflow Graphs

Scenario Aware Dataflow graphs [28] are a variant of dataflow models of computation that tries to occupy a sweet spot in the trade-off between analysability and expressiveness, in particular the ability to express more dynamic behaviour. It can be seen as a timed extension of the Heterochronous Dataflow model [16]. It also combines Synchronous Dataflow behaviour with finite state-automata. However, it extends the HDF model with time and optionally stochastic behaviour, by using Markov chains instead of FSMs. It further generalises the HDF model by allowing the FSM transitions to occur not only at the borders of complete iterations of the SDF behaviours, but also at intermediary stages. An important element of the timed model is that even though the FSM transitions occur in-between pieces of deterministic dataflow behavior, this does *not* mean that such pieces cannot overlap in time, i.e., cannot be pipelined. If that were not allowed, no tight performance predictions could be made. In this report, we exploit the fact that although they are pipelined, they are independent and their analysis can be sequentialized.

An important strength of the (timed) synchronous dataflow model is its determinism. An important goal of the SADF model is to mix a limited amount of non-deterministic behaviour by transitions of scenarios expressed by the FSM, without sacrificing the benefits of the deterministic behaviour within scenarios for efficient analysis.

The semantics of SADF can be captured by a combination of classical FSM semantics and $(\max,+)$-based semantics of the scenarios of determinate synchronous dataflow behaviour. The precise semantics is worked out in more detail in Section 6. The combination of state machines and $(\max,+)$-matrix multiplication is called a $(\max,+)$-automaton and is briefly introduced in the next subsection.

3.4 $(\max, +)$-automata

A $(\max, +)$-automaton [13], with some simplification of details not required for this report and rephrased using terminology of this report, is a tuple $\mathcal{A} = (\Sigma, \mathbf{M}, \mathcal{M})$, consisting of a finite set Σ of scenarios, a mapping $\mathbf{M}$, which assigns to every scenario $\sigma \in \Sigma$ a $(\max, +)$-matrix $\mathbf{M}(\sigma)$ and a morphism $\mathcal{M}$ on finite sequences of scenarios, mapping such sequences to a $(\max, +)$-matrix such that

$$\mathcal{M}(\sigma_1 \dots \sigma_k) = \mathbf{M}(\sigma_k) \dots \mathbf{M}(\sigma_1).$$

For a given sequence of scenarios, the automaton defines the completion time as follows:

$$\mathcal{A}(\sigma_1 \dots \sigma_k) = ||\mathcal{M}(\sigma_1 \dots \sigma_k)\mathbf{0}|| = ||\mathbf{M}(\sigma_k) \dots \mathbf{M}(\sigma_1)\mathbf{0}||.$$

Then, $\mathcal{M}(\bar{\sigma})\mathbf{0}$ captures the production times of the tokens of the SADF after the sequence $\bar{\sigma}$ of scenarios. We are typically interested in the time when the final token is produced. This is captured by taking the $(\max, +)$-norm (maximum entry) of the resulting vector. We are often interested in the worst-case throughput for any possible sequence of scenarios. This means, in the worst-case increase of $\mathcal{A}(\bar{\sigma})$ for growing length of $\bar{\sigma}$. Gaubert shows [13] how this maximum growth rate, determining minimum throughput can be efficiently computed as the maximum cycle mean of the equivalent timed event graph [1] of the matrix $\mathbf{M} = \max_{\sigma \in \Sigma} \mathbf{M}(\sigma)$ if the matrices are all square and of the same size. It also shows how, given an infinite regular sub language of Σ^*, the set of all finite scenario sequences, the maximum growth rate can be determined using a classical product automaton construction. Its worst-case behaviour can then by analysed using spectral analysis of a corresponding matrix, or if we are only interested in throughput by maximum cycle mean analysis directly on the automaton graph.

For this report, we will need some slight generalisations of this concept. In particular, instead of studying the average growth rate per *step* of the automaton, we will study the ratio of the growth rate relative to another quantity expressed as the sum of a certain benefit or *reward* per scenario. This amounts to application of the *generalised spectral problem* [9] to $(\max, +)$-automata. In this case, the worst-case throughput can be determined as an MCR of the automaton where edges now have two labellings, delays and rewards. Moreover, we will associate also non-square matrices $\mathbf{M}(\sigma)$ with scenarios. We assume that we always use a specification of legal scenario sequences that is consistent with the matrix sizes, i.e., such that the morphism $\mathcal{M}$ is well-defined. This can be achieved by creating duplicates of the states of the automaton for matrices of different size, which is also needed for the construction of the automaton product to restrict infinite scenario sequences to a regular sublanguage of Σ^ω.

4 A semantic model of weakly consistent SADF

In strongly consistent SADF graphs, every transition of the FSM corresponds to a *full iteration* of the SDF graph for the particular scenario. An iteration is the smallest collection of actor firings that returns the graph to its original state. It is therefore a piece of behaviour that can be repeated forever. Moreover, in this way it is clear that switches between scenarios are possible in such states, because the initial states are identical for all scenarios. *State* in this case refers to the tokens present in the graph, as the graph's actors and channels may

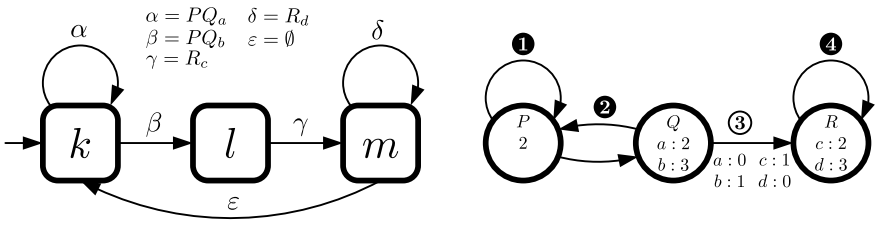


Figure 2: An example weakly-consistent SADF.

be different in different scenarios, but the initial tokens (having unique identities) are found in all scenarios. More specifically 'state' of those tokens refers to time stamps indicating the starting time of their availability to be further used (consumed), i.e., a dater function on the collection of initial tokens.

Although for many applications of this type of scenario aware dataflow behaviour, scenarios align well with pieces of behaviour that constitute iterations, there exist also situations in which it is convenient to consider units at smaller granularity, as explained above. We generalize the model to allow for the situation in which every edge of the FSM corresponds to an arbitrary (but fixed) collection of firings, not necessarily a single iteration or a whole number of iterations. This means that in contrast with the strongly consistent case, the starting state and ending state of a scenario graph are not necessarily the same and the definition of the possible sequences of scenarios needs to take into account that ending states of scenarios and starting states of the subsequent scenarios match. Practically, this means that they must have the same set of tokens. It is easy to see that all the firings along any cycle of the FSM must return the tokens to their original positions and that as long as the ending state of previous and starting states of the next scenario are consistent, then this constraint holds.

In order to establish a semantic framework for the behaviour of weakly consistent graphs we consider a generalisation of the semantics of strongly consistent graphs [15]. In that case every scenarios captures an iteration of an SDFG and this iteration can be fully and precisely characterised, in line with the $(\max, +)$ semantics of SDF, by multiplication a square $(\max, +)$-matrix. The size of the matrix is equal to the number of initial tokens in the graph. For weakly consistent graphs, this means that the number of the tokens before and after the execution of the firings of the scenario may be different, but nonetheless, this can be fully and precisely characterised by a $(\max, +)$-matrix. In this case however, the matrix need not be square. In the following section, we derive the semantic model for the running example graph.

5 Example

We establish the $(\max, +)$-automaton model of the running example graph shown in Figure 2. The initial state of the FSM is k and k has an edge to itself labelled with the scenario α, which corresponds to a firing of actor P and a firing of actor Q in 'mode' a. Initial tokens, shown as black dots at the edges, are labeled by numbers for reference. This combination of firings has no net effect on the distribution of the (three) tokens (Q_a produces no token to

position 3, in this scenario token 4 is untouched). Hence, it is consistent with starting and ending in the same FSM state k. Let's think of the starting state as being defined by the two tokens 1 and 2 in the figure, with their time stamps t_1 and t_2 respectively. This is captured in a $(\max, +)$-vector $[t_1 \;\; t_2]^T$. P needs to fire before Q_a and consumes both tokens. Hence, its earliest starting time is $\max(t_1, t_2)$. The firing takes 2 time units and completes at time $\max(t_1, t_2) + 2$, which in $(\max, +)$ sum-of-product form is equal to $\max(t_1 + 2, t_2 + 2)$, or in vector inner-product notation: $[2 \;\; 2] \cdot [t_1 \;\; t_2]^T$. This is the time stamp of the new token produced at position 1. Next, Q_a fires and consumes the token just produced by P on the edge from P to Q. Its firing takes also 2 units of time and completes at $\max(t_1 + 4, t_2 + 4)$, or in vector-product notation: $[4 \;\; 4] \cdot [t_1 \;\; t_2]^T$. This is the time at which the token at position 2 is reproduced. Combining the two symbolic states into a matrix-vector equation, we get the following relation between the starting state vector and the end state vector.

$$\begin{bmatrix} t'_1 \\ t'_2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

This matrix characterises the collective effect of the (two) firings in this scenario. Note however that, considering the whole graph, there is a third token present at position 4, that is neither consumed nor produced in this scenario. It is however part of the state and needs to be accounted for in state and matrix. It is easy to see that this is done by adding the following row and column.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -\infty \\ 4 & 4 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

The $(\max, +)$ neutral element, $-\infty$, of the max operator appears in those places where there is no dependency between certain tokens.

The transition $\alpha = PQ_a$ could be considered a complete iteration in the sense that it has no net effect on the location of the tokens, although actor R is not involved in the firings. This is not true for all edges however. Another edge, from state k to state l, is labeled with scenario β consisting of the firings $\{P, Q_b\}$. This is not a complete iteration as it produces an additional token on position 3 on the edge from Q to R. Along the lines of the previous scenario example and observing that the firing of Q takes 3 time units in mode b, we obtain the following matrix vector equation.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_3 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -\infty \\ 5 & 5 & -\infty \\ 5 & 5 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

Note that the matrix is no longer square, because the end state has four tokens while the starting state has three.

After this, a firing R_c will take place in scenario γ, moving from state l to m. This consumes tokens 3 and 4 and produces token 4, according to $t'_4 = [2 \; 2] \begin{bmatrix} t_3 \\ t_4 \end{bmatrix}$. $(t'_4 = \max(t_3 + 2, t_4 + 2))$. Token 3 disappears in this process and tokens 1 and 2 remain untouched. The full matrix thus has size 3 by 4.

From state m, an arbitrary number of scenarios 'δ' are possible, a single firing of R_d, each of which involves only token 4, according to $t'_4 = t_4 + 3$. The full matrix equation is as follows.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 3 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

At some point[1], a transition ϵ is taken, back to state k. It is labelled with an empty set of firings and therefore leaves all tokens at rest. The matrix representation would be a 3 by 3 identity matrix. Note that every cycle in the graph constitutes a collection of firings that has no net effect on the token distribution, i.e., is consistent. This makes the graph *weakly consistent*. As another a-priori sanity check on the graph, it can be verified by a straightforward reachability analysis whether or not a graph is deadlock free.

Equipped with the scenario matrices that characterise the effect on a system state of individual scenarios, we can determine the evolution of the system state for any given scenario sequence. Assume we have the following sequence $\alpha\alpha\beta\gamma\delta\delta\epsilon\alpha\beta\gamma$ and the initial state $\boldsymbol{t}_0 = [0 \;\; 0 \;\; 0]^T$ is such that all tokens are present at time $t = 0$. $\boldsymbol{t}_1$, the state after the first scenario α is: $\boldsymbol{t}_1 = \mathbf{M}(\alpha)\boldsymbol{t}_0 = [2 \;\; 4 \;\; 0]^T$. Continuing, the sequence of states evolves as follows.

$$\begin{aligned}
\boldsymbol{t}_1 &= \mathbf{M}(\alpha)\boldsymbol{t}_0 &&= [2 \;\; 4 \;\; 0]^T \\
\boldsymbol{t}_2 &= \mathbf{M}(\alpha)\boldsymbol{t}_1 &&= [6 \;\; 8 \;\; 0]^T \\
\boldsymbol{t}_3 &= \mathbf{M}(\beta)\boldsymbol{t}_2 &&= [10 \;\; 13 \;\; 13 \;\; 0]^T \\
\boldsymbol{t}_4 &= \mathbf{M}(\gamma)\boldsymbol{t}_3 &&= [10 \;\; 13 \;\; 15]^T \\
\boldsymbol{t}_5 &= \mathbf{M}(\delta)\boldsymbol{t}_4 &&= [10 \;\; 13 \;\; 18]^T \\
\boldsymbol{t}_6 &= \mathbf{M}(\delta)\boldsymbol{t}_5 &&= [10 \;\; 13 \;\; 21]^T \\
\boldsymbol{t}_7 &= \mathbf{M}(\epsilon)\boldsymbol{t}_6 &&= [10 \;\; 13 \;\; 21]^T \\
&\ldots
\end{aligned}$$

Figure 3 illustrates this process. Horizontally, a time line is shown. Vertically, the initial tokens t_i are shown. (Note that t_3 does not exist in all states.) Every scenario occurrence corresponds to a transformation on the state vector, amounting to multiplication by the corresponding matrix. The spaces between the state vectors are colored with a shade of grey corresponding to the scenario. The arrows have been added to illustrate in what logical order the scenarios occur and we would like to emphasise again that they can be taken separately, in sequential order, despite the fact that they overlap in time. This can be compared to the Gantt chart of the actual actor firings that are being modeled, shown below the graph. Here, the actor firings are colored with the scenarios in which they occur.

An important observation for deriving a performance analysis method is that we can trace the critical dependencies that determine the earliest guaranteed completion time of the sequence to be 24, by token t_4. By analysing the dependencies we can see that this timing

[1]Note that we could add acceptance conditions such as Büchi conditions to the automaton to enforce this. For simplicity, we do not do this in this report, but addition is straightforward. Moreover, they typically have no impact on the worst-case performance. In Section 7 we will extend the example to enforce such a transition.

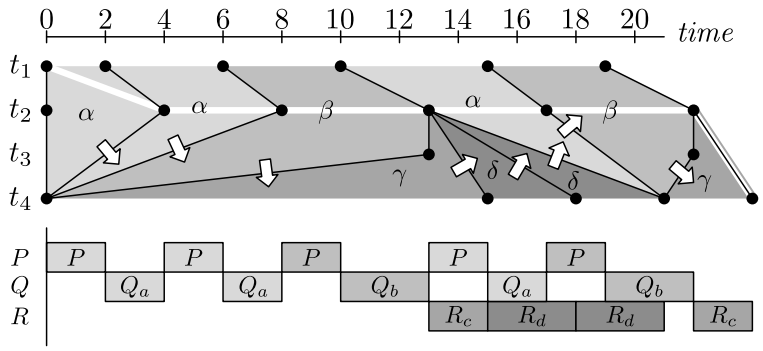


Figure 3: Execution of a sequence of scenarios

is determined by t_2 in the previous scenario, and as it turns it in this case for each earlier scenario up to the first where t_2 depended critically on t_1. The white line in the figure shows this dependency trace. In general, critical dependencies can be traced back through a path of *individual tokens*, independent of the state of other tokens in the vector.

6 Model and Semantics

We now formalize the model. We make precise how we specify the graph and what we mean by its throughput. A weakly consistent SADF graph is defined by a tuple $(\Sigma, G, \boldsymbol{\rho}, i, f, \pi, \mathcal{A})$. It has a finite set Σ of scenarios and every scenario $\sigma \in \Sigma$ has an associated SDF graph $G(\sigma)$ and a partial repetition vector $\boldsymbol{\rho}(\sigma)$, which maps every actor of $G(\sigma)$ to a non-negative number specifying how often the actor fires in the scenario. The graph $G(\sigma)$ has a collection of $i(\sigma) \in \mathbb{N}$ initial tokens, which are indexed $0 \leq n < i(\sigma)$. In the practical specification language, they are given names, but for the formalisation indices suffice. After execution of the partial repetition vector, the graph $G(\sigma)$ has a collection of $f(\sigma) \in \mathbb{N}$ 'final' tokens, which are indexed $0 \leq n < f(\sigma)$. (f can be determined from i, G and $\boldsymbol{\rho}$, but it is convenient to make it explicit.) We use the $(\max, +)$ semantics of the SDF graphs [1] to associate with every graph $G(\sigma)$, a $(\max, +)$-matrix $\mathbf{M}(G(\sigma)) \in (\mathbb{R}^{-\infty})^{f(\sigma) \times i(\sigma)}$, or in short $\mathbf{M}(\sigma)$, that precisely characterises the relationship between the time stamps of the initial and final tokens in the graph in that scenario as illustrated in Section 5. The FSM $\mathcal{A}$ is a tuple (Q, q_0, δ) with a set Q of states, an initial state q_0 and a labelled transition relation $\delta \subseteq Q \times \Sigma \times Q$. The scenario labels in the edges must be consistent in the sense that for any state $q \in Q$, any incoming edge labelled with scenario σ_1 and outgoing edge labelled with scenario σ_2, $f(\sigma_1) = i(\sigma_2)$, i.e., the number of final and initial tokens of subsequent scenarios must match. (We implicitly assume the tokens with the same index to be coupled, its time stamp at the end of scenario σ_1 is the initial time stamp for scenario σ_2.) We denote this number of tokens for a state q: $n(q)$. $\mathcal{A}$ accepts the sequence $\bar{\sigma}$ of scenarios if and only if there exists a sequence $\bar{q}$ of states such that $\bar{q}(0) = q_0$ and for every $n \geq 0$, there exists an edge $(\bar{q}(n), \bar{\sigma}(n), \bar{q}(n+1)) \in \delta$.

With sequence $\bar{\sigma}$, we associate the timing behaviour, a sequence of $(\max, +)$-vectors, such

that $\boldsymbol{t}_0 = \mathbf{0}$ and for all $n \geq 0$, $\boldsymbol{t}_{n+1} = \mathbf{M}(\bar{\sigma}(n))\boldsymbol{t}_n$. In the above definition, we can now clearly recognise the structure of a $(\max, +)$-automaton.

It is important to recall that, as is common in the timed dataflow performance analysis [26], it may be assumed that the (per scenario) constant execution times given in the model are in fact upper bounds for the real system behaviour and may in reality be non-deterministically smaller. Execution times may vary in a realisation due to variations in workload caused by the concrete data being processed, or by influences from its environment, for instance the amount of interference from arbitration of shared resources and other tasks in the system. Monotonicity of timing behaviour in dataflow graphs (and hence also in SADF graphs) which essentially follows from monotonicity of the $(\max, +)$-operators, ensures that performance guarantees derived for the model are in fact also guaranteed for such implementations.

For synchronous dataflow analysis, it is common to quantify throughput by measuring the number of iterations per time unit. In our case, it depends on the model how much actual, 'real-world' progress is made per scenario. We therefore assume that we explicitly quantify the amount of progress per scenario. For instance, for the example graph, we may be primarily interested in the number of firings of actor R. In this case the progress is 1 for scenarios γ and δ and 0 for any other scenario. For the MP3 example we may count the number of completed audio frames, by assigning progress of 1 (frame) to the scenarios *ss*, *sl*, *ls*, *ll* and *mm*, and 0 to the others. In general, we define a *reward* function $\pi : \Sigma \to \mathbb{R}^{\geq 0}$, which quantifies the amount of progress per scenario σ as $\pi(\sigma)$.

The throughput obtained from a scenario sequence $\bar{\sigma}$ can hence be defined as follows.

$$\tau(\bar{\sigma}) = \lim_{k \to \infty} \sup \frac{\sum_{n=0}^{k-1} \pi(\bar{\sigma}(n))}{||\boldsymbol{t}_k||}$$

I.e., throughput is defined as the average amount of progress made per unit of time. (lim sup is used instead of an ordinary limit, because the limit may not be defined for certain irregular scenario sequences.) The primary analysis question we answer in this report is to determine the worst-case throughput of an SADF graph:

$$\tau = \inf_{\bar{\sigma} \in \mathcal{L}(\mathcal{A})} \tau(\bar{\sigma}).$$

We can define an explicit state space semantics of the model. The states of the state-space consist of pairs $(q, \boldsymbol{t})$ consisting of a state $q \in Q$ of the FSM and a *normalized* vector $\boldsymbol{t}$. The initial state is $(q_0, \mathbf{0})$. The transitions of the state-space are constructed as follows. For a state $(q, \boldsymbol{t})$, consider every outgoing edge (q, σ, q') of q in the FSM. Then the state-space has a labelled transition

$$\left((q, \boldsymbol{t}), ||\boldsymbol{u}|| - ||\boldsymbol{t}||, \pi(\sigma), (q', \boldsymbol{u}^{norm})\right),$$

where $\boldsymbol{u} = \mathbf{M}(\sigma)\boldsymbol{t}$. The transitions are decorated with two labels. One signifies the amount of time progress in the scenario and the other the reward, the amount of progress made. It is not hard to show that if the graph is bounded (cannot accumulate an unbounded number of tokens on any edge) and it has integer (rational) execution times, then the state-space is finite.

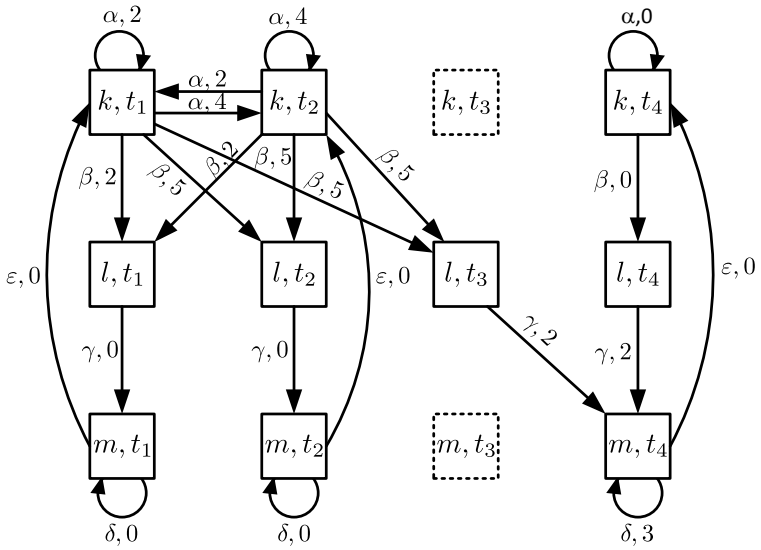


Figure 4: $(\max, +)$-automaton of the example graph.

7 Analysis

In order to determine the infimum of the throughput values for all possible scenario sequences on the graph, we need to find the worst-case scenario sequence. In any scenario sequence, both time and total reward progress with the scenarios being executed. Progress of time is measured as the $(\max, +)$ norm of the state vectors $\boldsymbol{t}_k$, i.e., the maximum element of the vector. Since $\boldsymbol{t}_{k+1} = \mathbf{M}(\bar{\sigma}(k))\boldsymbol{t}_k$, every element of $\boldsymbol{t}_{k+1}$ is determined by some element of $\boldsymbol{t}_k$ and offset by the corresponding dependency in the $(\max, +)$-matrix. This element in $\boldsymbol{t}_k$ can in turn be traced back to a single element in $\boldsymbol{t}_{k-1}$ and so forth back to $\boldsymbol{t}_0$. In other words, to study the relation between time progress and scenario sequences, we need not look at complete vectors, but we can concentrate on individual elements (initial / final tokens) and their individual dependencies as expressed by the entries in the matrices.

Figure 4 shows a structure which encodes these dependencies for the example of Figure 2. The nodes in this graph represent the initial/final tokens (horizontally) in each of the states of the FSM (vertically). For every edge of the FSM, we take the matrix $\mathbf{M}(\sigma)$, with σ the label of the edge, and for every finite (non $-\infty$) element in the matrix we draw an edge between the corresponding initial/final tokens and label it with the value of that element and with the reward $\pi(\sigma)$ of σ. For clarity we have labeled it with the scenario σ itself in the figure.

The precise definition of the $(\max, +)$-automaton corresponding to the SADF graph is as follows.

Definition 1 *For a given SADFG graph $(\Sigma, G, \boldsymbol{\rho}, i, f, \pi, \mathcal{A})$, the analysis $(\max, +)$-automaton is defined in the form of a graph (R, ϵ) with vertices R and edges ϵ, as follows.*

- $R = \{(q, i) \mid q \in Q, 1 \leq i \leq n(q)\}$
- $\epsilon = \{((q_1, i), \mathbf{M}(\sigma)_{i,j}, \pi(\sigma), (q_2, j)) \mid (q_1, \sigma, q_2) \in \delta, 1 \leq i \leq n(q_1), 1 \leq j \leq n(q_2)\}$

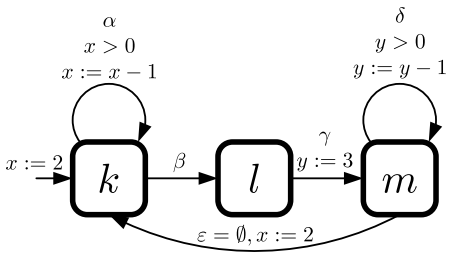


Figure 5: Refined example SADF.

The worst case throughput of the graph can be determined from a *maximum cycle ratio* analysis of the corresponding $(\max, +)$-automaton, i.e., find the simple cycle in the graph that represents the worst-case ratio of total reward over time progress.

Theorem 1 *Let* $\mathcal{G} = (\Sigma, G, \boldsymbol{\rho}, i, f, \pi, \mathcal{A})$ *be an SADF graph and* (R, ϵ) *be the corresponding* $(\max, +)$*-automaton graph, then* $\inf_{\bar{\sigma} \in \mathcal{L}(\mathcal{A})} \tau(\bar{\sigma}) = \mathrm{MCR}(R, \epsilon)$ *is the worst-case throughput of* $\mathcal{G}$*.*

Proof (Idea) Analogous to the results of Section VI of Gaubert [13], in particular Proposition 2, but generalised to the case of time progress divided by reward progress. □

Note that the example graph has cycles of zero reward (for instance the α self-loops) and hence the worst-case throughput is zero. Indeed, actor Q may never produce any output to R in which case, R will never fire. Let's refine the graph to limit the number of firings of Q in mode a to two, before it must fire in mode b. And let's similarly bound the number of firings of actor R in mode d to three.

This leads to the automaton shown in Figure 5, in which counters have been used to enforce the above constraints. After unfolding the counters to a plain FSM, it has 8 states. The corresponding $(\max, +)$-automaton then becomes the graph depicted in Figure 6. Counter x limits the number of subsequent α's to two, counter y the δ's to three.

We have analysed the graph of Figure 5 using the conversion and MCR analysis. The worst-case throughput is $\frac{1}{13}$. The critical cycle also tells us the scenario sequence that leads to this worst-case performance. In this case, it is determined to be the cycle of edges that is shown in bold in Figure 6. It corresponds to a repetition of the scenarios $\alpha\alpha\beta\gamma\epsilon$, which takes 13 units of time and involves only one firing of R.

8 Experimental Evaluation

We have implemented the worst-case throughput analysis method in the SDF^3 tool set [27] for analysis of dataflow models of computation as an extension to the available scenario-aware dataflow analysis. Our tool first parses an XML graph description of the model and extracts the individual scenario graphs. It then computes the $(\max, +)$-matrices corresponding to the scenarios by means of a symbolic execution of the corresponding dataflow graph. This

means that token time-stamps are symbolically represented as a $(\max, +)$ linear combination of the time-stamps of the initial tokens of the scenario as illustrated in Section 5. The sum-of-product expressions can be written as a vector inner-product of a vector of offsets and a vector with the initial time stamps. A matrix is extracted, where every row corresponds to a symbolic time-stamp represented as a row vector, taking the offset vectors. Finally, the $(\max, +)$-automaton is constructed from the scenario FSM and the scenario matrices as explained in Sections 5 and 7.

To get the performance analysis results, a maximum cycle ratio analysis is performed on the $(\max, +)$-automaton using the algorithm of Young et al. [34]. Alternatively one could use Howard's policy iteration [11, 17].

Our tool uses an XML based syntax specification language. We gave initial and final tokens specific names in the individual scenario specifications, so they can be correlated between scenarios without having to rely on index or ordering. This way we can also allow for the specification of initial tokens in a scenario graph that are not carried over from one scenario to the next. Upon entering such a scenario, they are initialised with a time-stamp of $-\infty$. We also added syntax to express the number of firings of each actor in a particular scenario, since they no longer execute according to complete iterations.

We have used the tool to analyse the MP3 decoder model of Figure 1. The specification consists of the seven dataflow graphs for the individual scenarios. The individual dataflow graphs only need to specify the actors involved in the scenario. The decoding (dec) scenario for instance, consists of a single actor, while the frame decoding graphs are fairly large (up to 25 actors for the mm scenario). The specified FSM has (after unfolding of the counters) 65 states. The $(\max, +)$-matrices extracted from the scenario dataflow graphs are 3 by 3 matrices, where the rows/columns represent one of the three processors on which the decoder is mapped. The determinate behaviour of the large scenario dataflow graphs, with many firings, can thus be very compactly represented. The $(\max, +)$-automaton that is constructed from the FSM and the matrices has 195 nodes (one for every combination of the three initial tokens and one of the 65 FSM states) and it has 2745 edges. The MCR analysis of this graph tells us the maximal throughput which is guaranteed to be attainable. The computation time on a standard PC is around 45ms. As a result we also get a critical scenario sequence from a critical cycle of the MCR analysis. For our example this is the sequence $ss \cdot (dec)^5 \cdot ss \cdot (dec)^4 \cdot rd \cdot dec$, in which we see that the worst case situation is that the decoder needs its maximum number of firings to produce an audio frame, combined with the (apparently) hardest of the frame synthesis scenarios, ss.

9 Conclusion

In this report we introduced an exact analysis method for a class of dynamic dataflow graphs, called weakly consistent scenario-aware dataflow in which the behaviour may non-deterministically vary according to scenarios of behaviour, yet within these scenarios behaviour is deterministic and follows the synchronous dataflow paradigm which provides us with powerful analysis techniques. We have defined an alternative semantics for the SADF model and we have shown how this semantics enables us to avoid mixing the non-deterministic behaviours and coherent dataflow behaviour despite their pipelined nature and overlap in

time. We have achieved this by finding an appropriate mapping of this type of graph to (a generalisation of) $(\max, +)$-automata and exploiting existing spectral analysis techniques in $(\max, +)$-algebra for performance analysis of our graphs. We have implemented the techniques in a tool for performance analysis of dataflow models and we see that it can effectively analyse a model of an MP3 decoder. Future work includes incorporating hierarchical state machines to model the possibly hierarchical detectors in the original SADF definition. Another interesting direction is to include stochastic information into the definition of scenario sequences by Markov chains and to analyse the system not only for worst-case, but also for long-run average or expected performance. We also want to explicitly apply the method for analysis of clusterings for quasi-static schedules. We also need to do more experimental evaluation for which we need to do more case-studies and develop the ability to generate random graphs, to investigate scalability.

References

[1] F. Baccelli, G. Cohen, G.J. Olsder, and J.P.Quadrat. *Synchronization and Linearity*. John Wiley & Sons, 1992.

[2] B. Bhattacharya and S.S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, October 2001.

[3] S. S. Bhattacharyya, P. Murthy, and E. Lee. APGAN and RPMC: Complementary Heuristics for Translating DSP Block Diagrams into Efficient Software Implementations. *Journal of Design Automation for Embedded Systems*, January 1997.

[4] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on signal processing*, 44(2):397–408, February 1996.

[5] J.Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[6] J. Buck and E. A. Lee. *Advanced Topics in Dataflow Computing and Multithreading*, chapter The Token Flow Model. IEEE Computer Society Press, 1994.

[7] Joseph T. Buck. A dynamic dataflow model suitable for efficient mixed hardware and software implementations of dsp applications. In *Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 165–172, 1994.

[8] J. Campos, G. Chiola, J.M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 39(5):386 –401, May 1992.

[9] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M.M. Gettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc. of the IFAC Conference on System Structure and Control*, Nantes, July 1998.

[10] Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. Des. Autom. Electron. Syst.*, 9:385–418, October 2004.

[11] Vishesh Dhingra and Stéphane Gaubert. How to solve large scale deterministic games with mean payoff by policy iteration. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, valuetools '06, New York, NY, USA, 2006. ACM.

[12] Joachim Falk, Joachim Keinert, Christian Haubelt, J"urgen Teich, and Shuvra Bhattacharyya. A Generalized Static Data Flow Clustering Algorithm for MPSoC Scheduling of Multimedia Applications. In *EMSOFT'08: Proceedings of the 8th ACM international conference on Embedded software*, October 2008.

[13] S. Gaubert. Performance evaluation of (max, +) automata. *IEEE Trans. Automatic Control*, 40(12):2014–2025, 1995.

[14] S. Gaubert and J. Mairesse. Modeling and analysis of timed petri nets using heaps of pieces. *IEEE Trans. Automatic Control*, 44(4):683–697, April 1999.

[15] M.C.W. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS 10, Proc., Scottsdale, Az, USA, 24-29 October, 2010*, pages 125–134, 2010.

[16] A. Girault, B. Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 18(6):742–760, June 1999.

[17] Ronald A Howard. *Dynamic Programming and Markov Processes*, volume 3. MIT Press, 1960.

[18] G. Kahn. The semantics of a simple language for parallel programming. In J.L. Rosenfeld, editor, *Information Processing 74: Proceedings of the IFIP Congress 74, Stockholm, Sweden, August 1974*, pages 471–475. North-Holland, Amsterdam, Netherlands, 1974.

[19] G. Kahn and D.B. MacQueen. Coroutines and networks of parallel programming. In B. Gilchrist, editor, *Information Processing 77: Proceedings of the IFIP Congress 77, Toronto, Canada, August 8-12, 1977*, pages 993–998. North-Holland, 1977.

[20] Richard M. Karp and Raymond E. Miller. Properties of a model for parallel computations: Determinancy, termination, queueing. *SIAM Jounal of Applied Mathematics*, 14(6):1390–1411, November 1966.

[21] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *IEEE Proceedings*, 75(9):1235–1245, September 1987.

[22] Edward A. Lee and Eleftherios Matsikoudis. *The Semantics of Dataflow with Firing.* Cambridge University Press, preprint (march 7 2 edition, 2007. Chapter from "From Semantics to Computer Science: Essays in memory of Gilles Kahn".

[23] Sjoerd Meijer, Hristo Nikolov, and Todor Stefanov. Throughput modeling to evaluate process merging transformations in polyhedral process networks. In *DATE*, pages 747–752, 2010.

[24] T Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[25] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.

[26] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., New York, NY, USA, 2009.

[27] S. Stuijk, M.C.W. Geilen, and T. Basten. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, pages 276–278. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006.

[28] Bart D. Theelen, Marc Geilen, Twan Basten, Jeroen Voeten, Stefan Valentin Gheorghita, and Sander Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *MEMOCODE*, pages 185–194, 2006.

[29] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, March 2000.

[30] Lothar Thiele and Nikolay Stoimenov. Modular performance analysis of cyclic dataflow graphs. In *EMSOFT '09: Proceedings of the seventh ACM international conference on Embedded software*, pages 127–136, New York, NY, USA, 2009. ACM.

[31] Stavros Tripakis, Dai Bui, Marc Geilen, Bert Rodiers, and Edward A. Lee. Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs. Technical Report UCB/EECS-2010-52, EECS Department, University of California, Berkeley, May 2010.

[32] G.X. Viennot. *Combinatoire Énumérative*, chapter Heaps of Pieces, I: Basic definitions and combinatorial lemmas, pages 321–350. Springer, 1986.

[33] Maarten H. Wiggers, Marco J.G. Bekooij, and Gerard J.M. Smit. Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS'08*, pages 183–194, April 2008.

[34] Neal E. Young, Robert Tarjan, and James Orlin. Faster parametric shortest path and minimum balance algorithms. *Networks*, 21(2):205–221, 1991.

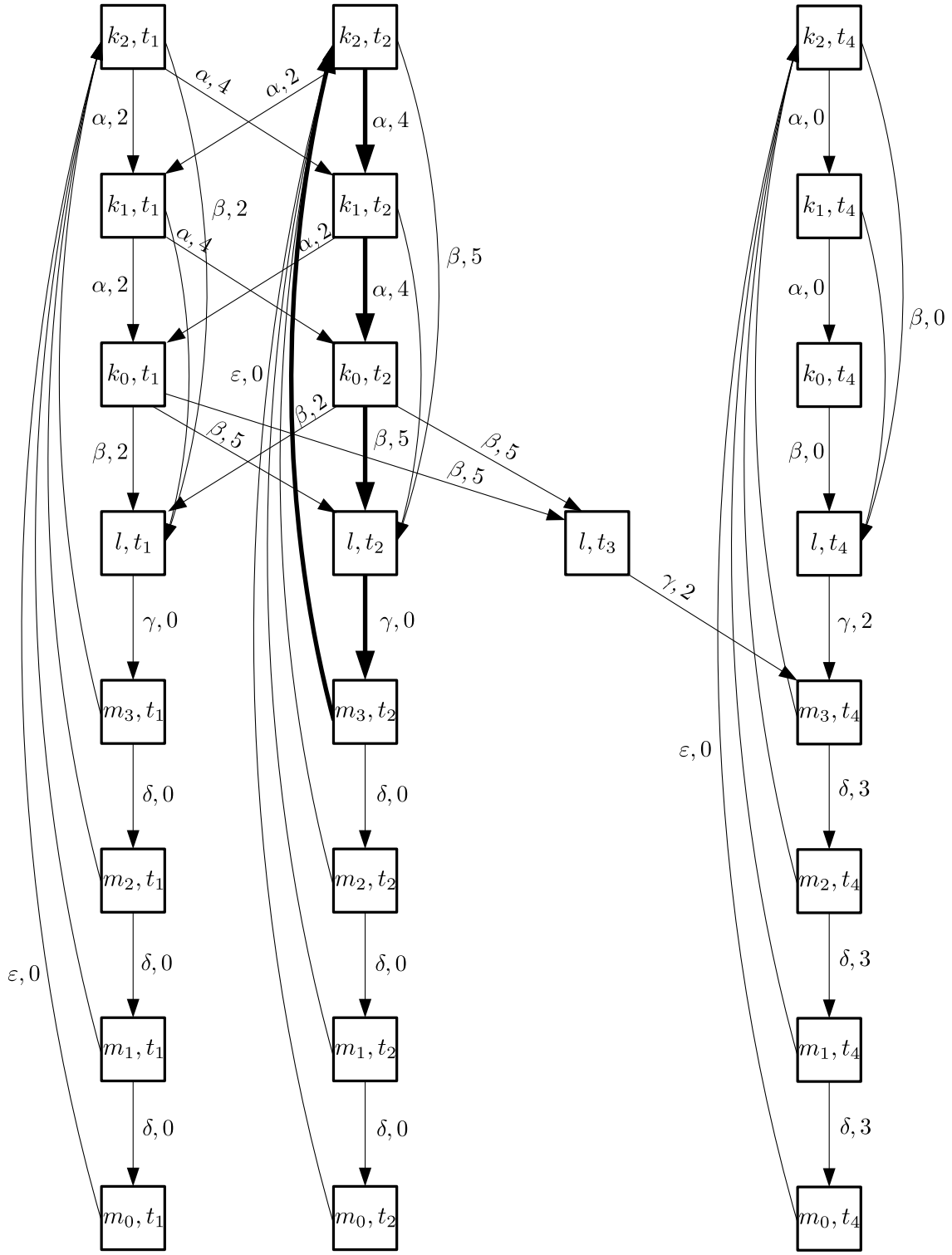


Figure 6: Extended $(\max, +)$-automaton of the example graph.