Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication

Maarten H. Wiggers [1], Marco J. G. Bekooij [2] and Gerard J. M. Smit [1]
[1] University of Twente, Enschede, The Netherlands
[2] NXP Semiconductors, Eindhoven, The Netherlands
m.h.wiggers@utwente.nl

Abstract

Streaming applications are often implemented as task graphs, in which data is communicated from task to task over buffers. Currently, techniques exist to compute buffer capacities that guarantee satisfaction of the throughput constraint if the amount of data produced and consumed by the tasks is known at design-time. However, applications such as audio and video decoders have tasks that produce and consume an amount of data that depends on the decoded stream. This paper introduces a dataflow model that allows for data-dependent communication, together with an algorithm that computes buffer capacities that guarantee satisfaction of a throughput constraint. The applicability of this algorithm is demonstrated by computing buffer capacities for an H.263 video decoder.

1. Introduction

Applications that process streams of data are often mapped on multi-processor systems for performance and power reasons. These applications include, for instance, audio/video decoding and post-processing. These streaming applications often have firm real-time requirements, such as throughput and latency constraints, of which throughput constraints dominate.

Typically, these applications are implemented as task graphs, in which data is processed by tasks and communicated over buffers. In our system, tasks only start their execution when there is sufficient data in all input buffers and sufficient space in all output buffers. This execution condition provides a robust mechanism against buffer overflow, but leads to a situation in which buffer capacities influence the throughput of the task graph.

For task graphs in which each task has a fixed execution condition that is known before execution of the graph, techniques exist to compute buffer capacities that are sufficient to guarantee satisfaction of the throughput constraint [14, 16, 19]. However, if tasks have an execution condition that can change in every execution depending on the processed data, then, currently, no techniques exist to compute buffer capacities that guarantee satisfaction of the throughput constraint.

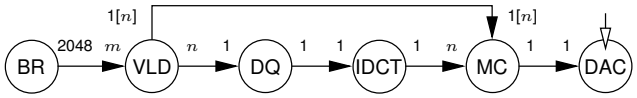

Figure 1. H.263 task graph.

The task graph of an H.263 video decoder, as shown in Figure 1, has data-dependent production and consumption quanta. In this task graph, we have a block reader (BR), a variable-length decoder (VLD), a dequantiser (DQ), an inverse discrete cosine transform (IDCT), a motion compensator (MC) and a digital-to-analog converter (DAC) task. The BR task reads blocks from a compact-disc. The number of bytes consumed, m, by the VLD task and the number of blocks produced, n, by the VLD task are both data-dependent. In order to enable the MC task to assemble a picture, the VLD communicates the number of blocks, $1[n]$, to the MC task. The DAC task is required to execute strictly periodically such that once every period a picture is displayed.

The contribution of this paper is a dataflow model that allows for data-dependent production and consumption quanta together with an algorithm that computes buffer capacities that are sufficient to guarantee satisfaction of a throughput constraint.

Our approach to compute buffer capacities for these dataflow graphs is as follows. First, we check whether the given graph is a valid input for our buffer capacity algorithm. Then we compute the maximum execution rate of each task as required to let the task on which the application places a throughput requirement execute strictly periodically. These execution rates can be computed, because a finite interval of values is associated with every parameter, which allows us to select parameter values that maximise

1080-1812/08 $25.00 © 2008 IEEE
DOI 10.1109/RTAS.2008.10

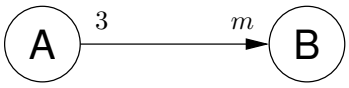

Figure 2. Task graph example, where m can attain a value from the set $\{2, 3\}$.

the execution rates. The maximum execution rate together with the selected parameter value result in a maximum data transfer rate on each buffer. This maximum transfer rate is taken as the rate of a linear bound on token transfer times on that buffer. Using these linear bounds, we compute for each buffer a minimum difference between the start times of the tasks such that on this buffer data is produced before it is consumed. These differences form the constraints in a min-cost max-flow formulation that results in start times that satisfy these constraints on all buffers and minimise the differences in start times, which minimises the required buffer capacities. We will be able to show that these buffer capacities are also sufficient to guarantee satisfaction of the throughput constraint for all other possible data transfer rates.

The task graph that is shown in Figure 2 illustrates that computing buffer capacities in case of data-dependent inter-task communication has its peculiarities. In this task graph, task A produces three data items in every execution and task B consumes either two or three data items in every execution. In case the consumption quantum equals three in every task execution, the minimum buffer capacity for deadlock-free execution is three. However, if the consumption quantum equals two in every task execution, then the minimum buffer capacity for deadlock-free execution is four. This example shows that maximising the consumption quantum does not lead to buffer capacities that are sufficient for other consumption quanta.

The outline of this paper is as follows. Section 2 discusses related work. Subsequently, we present our application model in Section 3, our analysis model in Section 4, and the rules to construct an analysis model from an application model in Section 5. In Section 6, we discuss in which cases we allow tasks to communicate the values of parameters. The algorithm to compute buffer capacities is presented in Section 7, while this algorithm is applied to an H.263 video decoder in Section 8. Finally, we conclude in Section 9.

2. Related Work

Related work can be split up in work that applies quasi static-order scheduling and work that applies run-time arbitration.

Approaches that construct quasi static-order schedules [2, 3, 5, 12] require the existence of a bounded length schedule for the (sub)graph. This requires that changes in production and consumption quanta only occur every (sub)graph iteration. However, for instance, a variable length decoder can change its consumption quantum in every execution. Furthermore, these approaches do not allow for pipelining production or consumption parameter changes, i.e. they do not allow that part of the (sub)graph already has a different value for a production or consumption parameter. The approach presented in [15], requires to change the implementation to a task graph that has constant production and consumption quanta, i.e. a constant number of produced or consumed data structures, but a variable size of the communicated data structure. In Section 8, we will show that their approach leads to larger buffer capacity requirements than our approach.

Current approaches that apply run-time arbitration characterise traffic [6, 7, 11] and require that there is a bounded difference between the upper and lower bounds on traffic, which implies that production and consumption quanta can change in every execution, but there has to be a fixed number of executions over which the amount of productions and consumptions is constant.

Our approach also applies run-time arbitration in order to allow tasks to change their productions and consumptions in every execution, but our approach does not require knowledge about the maximum difference between the upper and lower bound on transfers.

Cyclo-dynamic dataflow [17] and bounded dynamic dataflow [13] also apply run-time arbitration to allow for data-dependent execution rates, but do not provide an approach to calculate buffer capacities that guarantee satisfaction of a throughput constraint.

Another aspect that is different from related work is the following. We allow parameters to attain the value zero, which models conditional execution of tasks. Existing work that allows conditional execution of tasks, however, has its drawbacks. For boolean dataflow [3] graphs, we know that a consistent graph can still require unbounded memory. For well-behaved dataflow [4], we know that any graph constructed using the presented rules only requires bounded memory. However, no procedure is given that decides whether any given – boolean – dataflow graph is a well-behaved dataflow graph.

In contrast to existing work, we present a simple decision procedure to check whether any given task graph is a valid input for the algorithm that computes buffer capacities that satisfy a throughput constraint.

3. Task Graph

We first define a task graph and analysis model (dataflow graph) that only allow parameters that are local to tasks and do not support communication of parameter values between tasks. This is because, in order to define the constraints we place on parameter value communication in a precise yet

intuitive way, we require concepts from both task graphs and dataflow.

We assume that an application is implemented as a task graph. A task graph is a weakly connected directed graph $T = (W, B, S, \zeta, \eta, \kappa, \chi, \xi, \lambda)$. A weakly connected directed graph is a graph for which the underlying undirected graph is connected. We require that the throughput requirement of the application is placed on a task that either does not have any input buffers or does not have any output buffers. In such a task graph we have that tasks w_a and w_b, with $w_a, w_b \in W$, can communicate over a cyclic buffer $b_{ab} \in B$. Let b_{ab} denote a buffer over which task w_a sends data to task w_b. We say that tasks consume and produce containers on these cyclic buffers, where a container is a place-holder for data and all containers in a buffer have a fixed size. Tasks only start an execution when the previous execution has finished and there are sufficient full containers on their input buffers and sufficient empty containers on their output buffers such that the execution can finish without further waiting on container arrivals. The number of full containers that a task w_b requires on buffer $b_{ab} \in B$ is parameterised and given by $\lambda(b_{ab})$. The set of parameters is given by S. We define $\lambda : B \rightarrow S$, which is the function that returns the parameterised container consumption quantum on a particular buffer. Similarly, the number of full containers that a task w_a produces on buffer $b_{ab} \in B$, which equals the number of empty containers that are required on this buffer, is given by $\xi(b_{ab})$, with $\xi : B \rightarrow S$. With each parameter $s \in S$ a finite set of integer values is associated by $\chi(s)$, where we define $\chi : S \rightarrow \mathcal{P}_f(\mathbb{N})$. We let $\mathbb{N}$ denote the set of non-negative integer values, and we let $\mathcal{P}_f(\mathbb{N})$ denote the set of all finite subsets of $\mathbb{N}$ excluding the empty subset and the set only consisting of the value zero. Every execution, a task can change the number of containers that it consumes and produces. The worst-case response time is defined as the maximum difference between the time at which sufficient containers are present to enable an execution of task w_a and the time at which this execution finishes. The worst-case response time of task w_a is denoted by $\kappa(w_a)$, with $\kappa : W \rightarrow \mathbb{R}^+$. As in [20], we allow tasks to be scheduled at run-time by arbiters that can guarantee a worst-case response time given the worst-case execution times and the scheduler settings, i.e. the guarantee is independent of the rate with which tasks start their execution. This class of schedulers, for instance, includes time-division multiplex and round-robin. The capacity of a cyclic buffer b is given by $\zeta(b)$, with $\zeta : B \rightarrow \mathbb{N}$, while the number of initially filled containers is given by $\eta(b)$, with $\eta : B \rightarrow \mathbb{N}$.

4. Analysis Model

Our analysis model is Variable-Rate Dataflow (VRDF). A VRDF graph $G = (V, E, P, \delta, \rho, \phi, \pi, \gamma)$ is a directed graph that consists of a finite set of actors V and a finite set of edges E. A firing of an actor is enabled when on all input edges of the actor sufficient tokens are present. The number of tokens that are required on an edge $e \in E$ in a particular firing is parameterised, and given by $\gamma(e)$. The set of parameters is given by P. We define $\gamma : E \rightarrow P$, which gives the parameterised number of tokens consumed in any firing, i.e. the parameterised token consumption quantum, on a particular edge. Similarly, the parameterised number of tokens produced in any firing, i.e. the parameterised token production quantum, is given by $\pi(e)$, where we define $\pi : E \rightarrow P$. With each parameter $p \in P$ a finite set of integer values is associated by $\phi(p)$, where we define $\phi : P \rightarrow \mathcal{P}_f(\mathbb{N})$. Here we define $\mathcal{P}_f(\mathbb{N})$ as the set of all finite subsets of the non-negative integers excluding the empty set and the set containing only the value zero. The number of initial tokens on edge e is given by $\delta(e)$, with $\delta : E \rightarrow \mathbb{N}$, while the response time of an actor $v \in V$ is given by $\rho(v)$, with $\rho : V \rightarrow \mathbb{R}^+$. An actor v consumes its tokens in an atomic action at the start of a firing and produces its tokens in an atomic action $\rho(v)$ later at the finish of the firing. An actor does not start a firing before every previous firing has finished.

We define the following convenience functions. For an actor v_i, the function $E(v_i)$ provides the set of edges adjacent to v_i, while for a parameter p the function $E(p)$ provides the set of edges with token transfer quanta that are equal to p. Furthermore, for an actor v_i, the function $P(v_i)$ provides the union of the set of parameters that are token production quanta on edges from v_i and the set of parameters that are token consumption quanta on edges to v_i.

consistency On each edge of a VRDF graph the parameterised production and consumption quanta specify a relation between the execution rates of the two adjacent actors. If there are two directed paths that connect a given pair of actors and that specify inconsistent relations between the execution rates of this pair of actors, then either for any finite number of initial tokens this sub-graph will deadlock or there is an unbounded accumulation of tokens. Therefore, in order to verify whether there exists a bounded number of initial tokens that is sufficient to satisfy the throughput constraint, we need to check whether the relation between the execution rates of each pair of actors is strongly consistent [1, 8] over all paths between these two actors. We define strong consistency as the requirement that the constraints on execution rates are consistent for every valuation of the token transfer parameters.

On an edge e from actor v_a to actor v_b, we have the requirement that $q_a \cdot \pi(e) = q_b \cdot \gamma(e)$ should hold, where actor v_a fires proportionally q_a times and actor v_b fires proportionally q_b times. Similarly to [8], we collect all these requirements in matrix notation, i.e. we require a non-trivial

(symbolic) solution to exist for $\Psi \mathbf{q} = \mathbf{0}$, to verify whether a VRDF graph is strongly consistent. The matrix Ψ is a $|E| \times |V|$ matrix, where

$$\Psi_{ij} = \begin{cases} \pi(e_i) & \text{if } e_i = (v_j, v_k) \\ -\gamma(e_i) & \text{if } e_i = (v_k, v_j) \\ \pi(e_i) - \gamma(e_i) & \text{if } e_i = (v_j, v_j) \\ 0 & \text{otherwise} \end{cases}$$

In the matrix Ψ, each parameter p that can only attain a single value, i.e. $|\phi(p)| = 1$, is substituted by this value.

Definition 1 (Monotonic execution in the start times) *A dataflow graph executes monotonically in the start times if no decrease Δ in the start time of any firing can lead to an increase in the start time of any other firing.*

A VRDF graph executes monotonically in the start times. This is because the firing rules and token production rules of a firing are independent of the start time of the firing. Therefore, if a firing starts earlier, then tokens will only be produced earlier, which only leads to an earlier enabling and start of other actors.

Definition 2 (Linear execution in the start times) *A dataflow graph has linear temporal behaviour if a delay Δ in the start times cannot lead to delay larger than Δ for any start time of any firing.*

A VRDF graph has linear temporal behaviour, because a start time that is delayed by Δ can only lead to token productions that are delayed by maximally Δ. These tokens can delay another start time by maximally Δ, and so on.

The functional behaviour of VRDF graphs is deterministic in the sense that it is schedule independent, because the firing rule is sequential [10], i.e. production and consumption quanta are selected independently of the token arrival times.

5. Construction of Analysis Model

We construct a VRDF graph $G = (V, E, P, \delta, \rho, \phi, \pi, \gamma)$ from a task graph $T = (W, B, S, \zeta, \eta, \kappa, \chi, \xi, \lambda)$ as follows. Every task $w \in W$ is modelled by an actor $v \in V$, where the response time of the actor equals the worst-case response time of the task, i.e. $\rho(v) = \kappa(w)$. A buffer $b_{ab} \in B$ from task w_a to task w_b is modelled by two edges in opposite direction between the actors that model the tasks, i.e. edges $e_{ab}, e_{ba} \in E$ are added if v_a models w_a and v_b models w_b. The number of initial tokens on edge e_{ab} corresponds with the number of initially filled containers on buffer b_{ab}, i.e. $\delta(e_{ab}) = \eta(b_{ab})$. The number of tokens on edge e_{ba} corresponds with the remaining initial containers on b_{ab}, i.e. $\delta(e_{ba}) = \zeta(b_{ab}) - \eta(b_{ab})$.

Every parameter $s \in S$ is modelled by a parameter $p \in P$ that has the same set of values associated with it, i.e. $\phi(p) = \chi(s)$. Given that e_{ab} and e_{ba} together model buffer b_{ab}. If the number of containers produced on buffer b_{ab} equals s, then the number of tokens produced on edge e_{ab} in the VRDF graph equals p, where p models s. In every firing, the number of tokens consumed from edge e_{ba} equals the number of tokens produced on edge e_{ab}, i.e. $\gamma(e_{ba}) = \pi(e_{ab})$. The number of tokens consumed per firing from edge e_{ba} models the number of empty containers that are required for task w_a to start. Further, we have that if the number of containers consumed from buffer b_{ab} equals s', then the number of tokens consumed from edge e_{ab} equals p', where p' models s'. In every firing, the number of tokens produced on edge e_{ba} equals the number of tokens consumed from edge e_{ab}, i.e. $\pi(e_{ba}) = \gamma(e_{ab})$. Throughout this paper, actor v_τ models the task of which the application requires a strictly periodic execution with period τ.

6. Parameter distribution

In principle, it is required that every parameter is only associated with a single task. The only exception is that we do allow for every parameter $s \in S$ one buffer b_{ab} over which the values of parameter s are communicated to another task. In Figure 3 buffer b_{ab} communicates the value of parameter s. Of this buffer b_{ab}, we require that the container production and consumption quanta are constant and equal to one, i.e. $\xi(b_{ab}) = \lambda(b_{ab}) = 1$, and that there are no initially filled containers on this buffer, i.e. $\eta(b_{ab}) = 0$. Furthermore we require that this buffer is from a task w_a that does not have any container consumption quantum that is parameterised in s to a task w_b that does not have any container production quantum that is parameterised in s.

This construct allows both tasks to have container transfer quanta that are parameterised in parameter s, as for example shown in task graph T_s of Figure 3 on the edges to and from the sub-graph T'_s. For this purpose we introduce the function ψ, which returns the parameter value that is communicated over a buffer. The function ψ is defined as $\psi : B \rightarrow S \cup \{\varepsilon\}$, where ε is an undefined parameter value. Similarly, we introduce the function φ, which returns the parameter value that is communicated over an edge. The function φ is defined as $\varphi : E \rightarrow P \cup \{\epsilon\}$, where ϵ is an undefined parameter value. Given a buffer b_{ab} from w_a to w_b with $\psi(b) = n$ and given actor v_a that models w_a, actor v_b that models w_b, edges e_{ab} and e_{ba} that model buffer b_{ab}, and parameter p that corresponds with parameter s. Then $\varphi(e_{ab}) = p$ and $\varphi(e_{ba}) = \epsilon$. In the task graph of Figure 3, the annotation at the end points of an edge denote the paremeterised container transfer quantum. In case that the value of a parameter is transferred, then this is denoted by a parameter between square brackets after the container transfer quantum, which is required to be one. Communication of undefined values is not depicted in the task graphs or dataflow graphs.

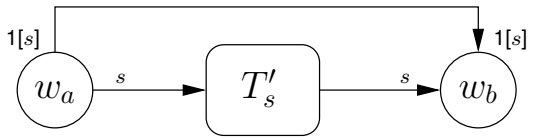

Figure 3. Task graph T_s and sub graph T'_s, with, for instance, $\chi(s) = \{0, 1\}$.

We know that a strongly consistent Boolean Dataflow graph does not always execute in bounded memory [3]. By placing the following restriction we only allow VRDF graphs for which strong consistency implies execution in bounded memory. Informally stated, the restriction is that the repetition vector of the VRDF graph G_p that models task graph T_s of Figure 3 has repetition rates for actors v_a and v_b that are both equal to one, i.e. $q_a = q_b = 1$. This enforces a clear relation between the parameter values on edge e_{ab} and the tokens on paths from v_a to v_b through VRDF sub-graph G'_p, which models T'_s, since every firing of v_a produces one token on e_{ab} enabling one execution of v_b and also produces just enough tokens on the paths through G'_p to enable one execution of v_b. After v_b has fired we have that on all edges tokens have returned to the edges on which they were initially placed. The tokens on e_{ab} that contain parameter values thus each relate to an iteration of G_p. A situation that is not allowed for the graph shown in Figure 3 is the situation in which T'_n consists of a task w_x that produces and consumes two containers in every firing, resulting in $q_a = q_b = 2$ and $q_x = n$ in the VRDF graph G_p. This is because s could first be equal to one and could subsequently be zero for an unbounded number of executions of w_a, leading to an unbounded accumulation of containers on buffer b_{ab}.

More formally stated, the restriction for every parameter s that is a container production quantum of task w_a and a container consumption quantum of w_b is as follows. Let v_a model w_a, let v_b model w_b and let p model s. Then we define sub-graph G_p as the graph that includes all simple directed paths from v_a to v_b and from v_b to v_a of which each such simple directed path includes edges from $E(p)$. It is required that the repetition rate of v_a and v_b in the graph G_p equals one.

In a strongly consistent and strongly connected VRDF graph, we can construct G_p as follows. First we check that for every parameter p there is not more than one edge in the VRDF graph that communicates the values of parameter p. Given such an edge e from v_a to v_b, we know that v_a sends the values of parameter p to v_b. Now, we need to find the sub-graph G'_p formed by the set of actors V'_p and edges E'_p that are on a simple directed path from v_a to v_b or vice versa, where these paths start with an edge that has a token production quantum parameterised in p and end with an edge that has a token consumption quantum that is parameterised in p. Since all actors in V'_p have a simple directed path to v_a and v_b that contains a token transfer quantum that is parameterised in p, consistency tells us that the actors in V'_p do not have a simple directed path to v_a or v_b that does not have a token transfer quantum that is parameterised in p. This means that removal of all edges that have a token transfer quantum that is parameterised in p leads to disconnection of the actors V'_p from v_a and v_b. Since the sub-graph G'_p is still strongly connected, a breadth-first search from one of the actors in V'_p will find all actors in V'_p.

7. Buffer Capacities

In this section, we will first present an algorithm, which selects parameter values that maximise the execution rate of each actor relative to v_τ. From the relation between execution rates and the required period of v_τ, we derive for each actor the minimal time between subsequent starts. Subsequently, the selected parameter values and derived minimum time between subsequent starts result in a token transfer rate. This token transfer rate will be the slope of linear bounds on token production and consumption times. Given such a bound, we will define a schedule for that actor, which is thus specific for that actor on the edge under consideration. Given these linear bounds on each edge we can derive a minimum difference between the start times of the actors adjacent to this edge. For this difference holds that tokens are produced before they are consumed if this edge and its adjacent actors are considered in isolation. These differences form the constraints in a min-cost max-flow formulation that results in start times that satisfy these constraints and minimise the differences in start times, which minimises the required number of initial tokens. Subsequently, in Section 7.4, we will first show that for the selected parameter values, these schedules per edge are in fact all the same for each actor. Then we will show that for parameter values different from the selected parameter values the computed number of initial tokens is still sufficient to let v_τ execute strictly periodically.

7.1. Maximum Execution Rates

In this section we derive the token transfer quanta that lead to the maximum execution rate of each actor relative to the execution rate of actor v_τ.

Algorithm 1 results in a graph in which each actor needs to execute with the highest rate in order to let v_τ execute strictly periodically, while still only requiring a bounded number of initial tokens on every edge.

On any edge e_{ab}, we have the balance equation $q_a \cdot \pi(e_{ab}) = q_b \cdot \gamma(e_{ab})$. If, for every edge, the execution rates satisfy this equation then a bounded number of initial tokens is required. It is clear from the balance equation that q_a/q_b is maximised if $\gamma(e_{ab})/\pi(e_{ab})$ is maximised, which implies maximising $\gamma(e_{ab})$ and minimising $\pi(e_{ab})$. On edge

Algorithm 1 Determine quanta that result in maximum execution rate of each actor relative to v_τ.

1. compute the minimum distance in number of edges from each actor v_i to v_τ, i.e. $d(v_i)$, with a breadth-first search from v_τ

2. visit the actors in a breadth-first fashion from v_τ and for each $v_i \in V$ and for each $p \in P$ if $A(v_i, p) \wedge B(v_i, p)$ holds then $\bar{p} = \check{p}$, otherwise $\bar{p} = \hat{p}$

$$A(v_x, u) = \exists e \in E \bullet ((e = e_{xy} \wedge \pi(e_{xy}) = u) \vee (e = e_{yx} \wedge \gamma(e_{yx}) = u)) \wedge d(v_y) < d(v_x)$$
$$B(v_x, u) = \nexists v_y \in V \bullet d(v_y) < d(v_x) \wedge u \in P(v_y)$$

e_{ba}, we have that q_a/q_b is maximised if $\gamma(e_{ba})$ is minimised and $\pi(e_{ba})$ is maximised. This provides the rationale behind Algorithm 1, which takes the minimum value for a parameter local to v_i if it is used as a token transfer quantum on an edge e_i to or from an actor that is closer to v_τ than v_i, and takes the maximum value otherwise.

By solving the balance equations for the resulting valuation of the parameters, we can derive $\omega(v_i)$, which is the minimum time between subsequent starts of actor v_i. This is because we know that v_τ is required to fire with period τ and that any actor v_i maximally fires q_i/q_τ times as often as v_τ, while still executing in bounded memory. This means that $\omega(v_i) = q_\tau/q_i \cdot \tau$.

From now on, for any parameter p, let $\bar{p}$ denote the selected value for parameter p in the just described procedure that derives the maximum execution rates.

7.2. Minimum Start Time Differences

Given two actors v_a and v_b, and an edge e_{ab}. On edge e_{ab}, $\pi(e_{ab}) = m$, $\gamma(e_{ab}) = n$, with $m, n \in P$, and $\delta(e_{ab}) = d$. We define $\hat{\alpha}_p(x, e_{ab}, \sigma(v_a, e_{ab}))$ as the linear upper bound on the production time of token x on e_{ab} under schedule $\sigma(v_a, e_{ab})$, while $\check{\alpha}_c(x, e_{ab}, \sigma(v_b, e_{ab}))$ is the linear lower bound on the consumption time of token x on e_{ab} under schedule $\sigma(v_b, e_{ab})$. With $q_a \cdot \bar{m} = q_b \cdot \bar{n}$, we have, by construction, that $\omega(v_a)/\bar{m} = \omega(v_b)/\bar{n}$, which we take as the rate of both linear bounds.

Let $\Pi(f, e_{ab})$ be the cumulative number of tokens produced on edge e_{ab} in firings one up to and including firing f, with $f \in \mathbb{N}^*$ and $\Pi(0, e_{ab}) = \delta(e_{ab})$, where $\mathbb{N}^*$ is the set of positive integers. We define the start time of firing f in schedule $\sigma(v_a, e_{ab})$ for v_a on edge e_{ab} by

$$s(v_a, f, \sigma(v_a, e_{ab})) = s(v_a, 1, \sigma(v_a, e_{ab})) + \frac{\omega(v_a)}{\bar{m}}(\Pi(f-1, e_{ab}) - \delta(e_{ab})) \quad (1)$$

We require that all schedules defined for an actor have an equal start time for the first firing of this actor.

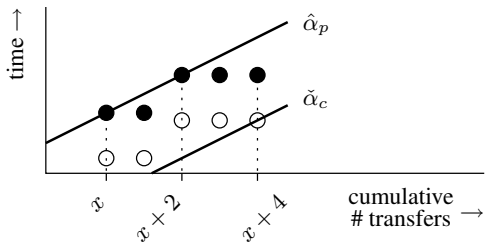

Figure 4. Token production and consumption schedules on one buffer, with parameter n and $\phi(n) = \{2, 3\}$.

This schedule implies that token $\Pi(f-1, e_{ab}) + 1$ up to and including token $\Pi(f, e_{ab})$ are produced at $s(v_a, f, \sigma(v_a, e_{ab})) + \rho(v_a)$. A linear upper bound on the production time of token x, with $x \in \mathbb{N}^*$, on edge e_{ab} with schedule $\sigma(v_a, e_{ab})$ is therefore

$$\hat{\alpha}_p(x, e_{ab}, \sigma(v_a, e_{ab})) = s(v_a, 1, \sigma(v_a, e_{ab})) + \rho(v_a) + \frac{\omega(v_a)}{\bar{m}}(x - \delta(e_{ab}) - 1) \quad (2)$$

As illustrated in Figure 4, this bound is exact for the production time of token $\Pi(f-1, e_{ab}) + 1$ given schedule $\sigma(v_a, e_{ab})$.

Let $\Gamma(f, e_{ab})$ be the cumulative number of tokens consumed from edge e_{ab} in firings one up to and including firing f, with $f \in \mathbb{N}^*$ and $\Gamma(0, e_{ab}) = 0$. We define the start time of firing f in schedule $\sigma(v_b, e_{ab})$ for v_b on edge e_{ab} by

$$s(v_b, f, \sigma(v_b, e_{ab})) = s(v_b, 1, \sigma(v_b, e_{ab})) + \frac{\omega(v_b)}{\bar{n}}\Gamma(f-1, e_{ab}) \quad (3)$$

This means that token $\Gamma(f-1, e_{ab})+1$ up to and including token $\Gamma(f, e_{ab})$ are consumed at $s(v_a, f, \sigma(v_a, e_{ab}))$. An upper bound on $\Gamma(f, e_{ab})$ is given by $\Gamma(f-1, e_{ab}) + \hat{\gamma}(e_{ab})$. A linear lower bound on the consumption time of token x, with $x \in \mathbb{N}^*$, from edge e_{ab} with schedule $\sigma(v_b, e_{ab})$ is therefore

$$\check{\alpha}_c(x, e_{ab}, \sigma(v_b, e_{ab})) = s(v_b, 1, \sigma(v_b, e_{ab})) + \frac{\omega(v_b)}{\bar{n}}(x - \hat{\gamma}(e_{ab})) \quad (4)$$

As illustrated in Figure 4, the bound is exact in case $\Gamma(f, e_{ab}) = \Gamma(f-1, e_{ab}) + \hat{\gamma}(e_{ab})$ given schedule $\sigma(v_b, e_{ab})$.

Since, on any edge, tokens can only be consumed after they have been produced, we have that for every token x, $\check{\alpha}_c(x, e_{ab}, \sigma(v_b, e_{ab})) \geq \hat{\alpha}_p(x, e_{ab}, \sigma(v_a, e_{ab}))$

should hold. After substitution of Equations (2) and (4) together with the knowledge that $\omega(v_a)/\bar{m} = \omega(v_b)/\bar{n}$, this implies that

$$s(v_b, 1, \sigma(v_b, e_{ab})) - s(v_a, 1, \sigma(v_a, e_{ab})) \geq \frac{\omega(v_a)}{\bar{m}}(\hat{\gamma}(e_{ab}) - \delta(e_{ab}) - 1) + \rho(v_a) \quad (5)$$

7.3. Network Flow Problem

Similar as in [19], we interpret the predefined number of initial tokens on an edge as a constraint, which influences the constraint stated in Equation (5) on the minimal difference in start times of the adjacent actors. These differences form the constraints in the network flow problem in Algorithm 2 that minimises the start times of all actors relative to the start time of a dummy-actor v_0. This implies minimising the differences in start times. If there is a solution that satisfies the constraints then a sufficient number of initial tokens can be computed with the resulting start times together with the bounds on token production and consumption times. This is done by rewriting Equation (5) to Equation (6). We take as number of initial tokens the smallest integer value that satisfies the constraint in Equation (6). Note that the predefined number of initial tokens were a constraint on the start times, and that given the start times computed in the network flow problem we now derive a, possibly smaller, number of tokens that is sufficient given these start times.

$$\delta(e_{ab}) \geq \hat{\gamma}(e_{ab}) - 1 + \frac{\bar{m}}{\omega(v_a)}\big(\rho(v_a) + s(v_a, 1, \sigma(v_a, e_{ab})) - s(v_b, 1, \sigma(v_b, e_{ab}))\big) \quad (6)$$

The start times of the first firings of each actor are computed as follows. We construct a graph G_0 from a VRDF graph G as follows. We extend the VRDF graph with an additional actor v_0, $V_0 = V \cup \{v_0\}$, and extend the set E with edges from v_0 to every actor in V. We define the valuation function $\beta : E \rightarrow \mathbb{R}$, where for edge $e_{ab} \in E$ we define

$$\beta(e_{ab}) = \frac{\omega(v_a)}{\bar{m}}(\hat{\gamma}(e_{ab}) - \delta(e_{ab}) - 1) + \rho(v_a) \quad (7)$$

while for every edge e adjacent to v_0 we have $\beta(e) = 0$. Subsequently, we solve the network flow problem in Algorithm 2.

7.4. Sufficiency of Buffer Capacities

In this section, Theorem 1 will provide us with a reference result that says that if all parameters p have value $\bar{p}$, then v_τ can execute strictly periodically. Subsequently, Lemmas 1, 2, 3, 4, and 5 will help in establishing Theorem 2, which states that v_τ can execute strictly periodically if there are no parameters shared by two actors. This section is concluded by Lemmas 6 and 7 that support the proof of Theorem 3, which states that v_τ can execute strictly periodically, even if parameters are shared by two actors, given the number of initial tokens as computed in the previous section. The discussion in this section assumes that each actor v_i has a constant response time. However, since a VRDF graph is monotonic in the start times and smaller response times can only lead to earlier token production times and earlier actor enabling times, the results of this section are also valid if $\rho(v_i)$ is an upper bound of the response time of actor v_i.

Algorithm 2 Start time computation

$$\min \sum_{v_i \in V} s(v_i)$$

subject to

$$s(v_j) - s(v_i) \geq \beta(e_{ij}) \qquad \forall e_{ij} \in E_0$$
$$s(v_0) = 0$$

Theorem 1 *If each parameter $p \in P$ has value $\bar{p}$, then the number of initial tokens computed with Equation* (6) *is sufficient to let v_τ execute strictly periodically with period τ.*

Proof. With constant parameter values a VRDF graph is a multi-rate dataflow graph [9]. The reasoning now follows [18]. With constant parameter values, the difference between subsequent start times in the constructed schedules is constant and equals $\omega(v_i)$. For each actor v_i, $\omega(v_i)$ is derived using the execution rates to translate the period of v_τ. Given for each actor a schedule in which the difference between subsequent starts is $\omega(v_i)$ and given parameter value $\bar{p}$, the linear bounds are conservative for the resulting token production and consumption times. Since Algorithm 2 computes start times given these bounds and Equation (6) gives a number of tokens based on these linear bounds, the resulting number of tokens is sufficient to enable for each actor v_i the schedule in which the difference between subsequent start times is $\omega(v_i)$. □

Lemma 1 states that in any schedule constructed for an actor on an edge, we have that the difference between subsequent starts is proportional to the token transfer quantum by that actor on that edge.

Lemma 1 *Given an actor v_i and a parameter $p \in P(v_i)$ that has value p_f in firing f of v_i. Then on all edges $e \in E(p)$ the difference between the starts of firings $f+1$ and f, with $f \in \mathbb{N}^*$, of v_i under schedule $\sigma(v_i, e)$ is*

$$\frac{\omega(v_i)}{\bar{p}} \cdot p_f \quad (8)$$

Proof. In case v_i produces on e, with $\pi(e) = p$ and a production quantum of p_f in firing f, then according to Equation (1), we have that

$$s(v, f+1, \sigma(v_i, e)) - s(v, f, \sigma(v_i, e)) = \frac{\omega(v_i)}{\bar{p}}(\Pi(f, e_i) - \Pi(f-1, e)) = \frac{\omega(v_i)}{\bar{p}} \cdot p_f \quad (9)$$

In case v_i consumes from e, with $\gamma(e) = p$ and a consumption quantum of p_f in firing f, then according to Equation (3), we have that

$$s(v, f+1, \sigma(v_i, e)) - s(v, f, \sigma(v_i, e)) = \frac{\omega(v_i)}{\bar{p}}(\Gamma(f, e) - \Gamma(f-1, e)) = \frac{\omega(v_i)}{\bar{p}} \cdot p_f \quad (10)$$

□

Definition 3 *For two schedules σ_1 and σ_2 that determine start times of v_i, we have $\sigma_1 \leq \sigma_2$ iff the start time of every firing f in σ_1 is not later than the start time of f in σ_2, i.e.*

$$\sigma_1 \leq \sigma_2 \Leftrightarrow \forall f \in \mathbb{N}^* \bullet s(v_i, f, \sigma_1) \leq s(v_i, f, \sigma_2) \quad (11)$$

Definition 4 *For every actor v_i and edge $e \in E(v_i)$, schedule $\sigma(v_i, e)$ is the constructed schedule of actor v_i on edge e, which can be linearly bounded and of which the start time of the first firing is computed in Algorithm 2.*

Definition 5 *For every actor v_i and edge $e \in E(v_i)$, schedule $\bar{\sigma}(v_i, e)$ is the schedule $\sigma(v_i, e)$ of actor v_i on edge e for parameter value $\bar{p}$, with p equal to the token transfer quantum of v_i on e.*

Definition 6 *For actor v_j and edge e_{ij}, schedule $\sigma(v_j, e_{ij})$ is valid, denoted $\mathsf{valid}(\sigma(v_j, e_{ij}))$, iff for every token $x \in \mathbb{N}^*$ the production time of token x on e_{ij} is earlier than or equal to the consumption time of token x on e_{ij}.*

Definition 7 *For every actor v_i, schedule $\sigma(v_i)$ is a constructed schedule of actor v_i given by*

$$\sigma(v_i) = \max(\{\sigma(v_i, e) | e \in E(v_i)\}) \quad (12)$$

The schedule that is defined by Equation (12) is a schedule that is constructed independently of token arrival times. Definition 8 defines when this constructed schedule is valid with respect to token arrival times.

Definition 8 *For actor v_j, schedule $\sigma(v_j)$ is valid, denoted $\mathsf{valid}(\sigma(v_j))$, iff for every token $x \in \mathbb{N}^*$ and every edge $e_{ij} \in E(v_j)$, the production time of token x on e_{ij} is earlier than or equal to the consumption time of token x on e_{ij}.*

In Definition 6, a constructed schedule on an input edge is defined valid if token consumption times are not earlier than token production times. Definition 9, defines the schedule on an output edge of an actor to be valid if (1) the constructed schedule of the actor equals the schedule on this output edge and (2) on all input edges of this actor tokens arrive in time to enable the constructed schedule of this actor.

Definition 9 *For actor v_i and edge e_{ij}, schedule $\sigma(v_i, e_{ij})$ is valid, denoted $\mathsf{valid}(\sigma(v_i, e_{ij}))$, iff $\sigma(v_i)$ is valid and $\sigma(v_i) = \sigma(v_i, e_{ij})$.*

Assumption 1 *Any parameter p that is shared by two actors has a constant value $\bar{p}$.*

$$\forall p \in P \; \exists v_i, v_j \in V \bullet v_i \neq v_j \implies p = \bar{p} \quad (13)$$

Under Assumption 1, we only have parameters that are local to an actor and parameters that are constants. Lemma 2 states that under this assumption, we have for any actor that if tokens arrive in time for all schedules constructed per edge, then tokens also arrive in time for the schedule constructed for the actor. This is a non-trivial result, because the schedule of the actor has later start times than the schedules constructed per edge, which implies that it produces tokens later.

Lemma 2 *Given Assumption 1. Then the following holds*

$$\forall v_j \in V \bullet \big(\forall e_{ij} \in E(v_j) \bullet \mathsf{valid}(\sigma(v_j, e_{ij}))\big) \implies \mathsf{valid}(\sigma(v_j)) \quad (14)$$

Proof. Equation (12) states that $\forall e \in E(v_j) \bullet \sigma(v_j) \geq \sigma(v_j, e)$, i.e. $\sigma(v_j)$ has token production times that are later than or equal to $\sigma(v_j, e)$.

Consistency tells us that there is no simple cycle that has only a single occurrence of a token transfer quantum that is parameterised in a non-constant parameter p. Given Assumption 1, this means that every directed path from v_j to v_j that starts with an edge $e_1 \in E(p)$ ends with an edge $e_2 \in E(p)$. Since all schedules constructed for an actor have an equal start time for the first firing of this actor, Lemma 1 tells us that $\sigma(v_j, e_1) = \sigma(v_j, e_2)$.

Similarly, by consistency, any directed path from v_j to v_j that starts with an edge e_1 on which v_j has a constant token production quantum ends with an edge e_2 on which v_j has a constant token consumption quantum. Since all schedules constructed for an actor have an equal start time for the first firing of this actor and the considered quanta are constant, Lemma 1 tells us that $\sigma(v_j, e_1) = \sigma(v_j, e_2)$.

Since a VRDF graph has linear temporal behaviour, a delay $\Delta = s(v_j, f, \sigma(v_j)) - s(v_j, f, \sigma(v_j, e_1))$ in token production times of firing f of v_j on an edge e_1 leads to

token arrival times on any corresponding edge e_2 that are maximally delayed by Δ. According to Definition 6, if $\mathsf{valid}(\sigma(v_j, e_2))$, then tokens arrive before the token consumption times that follow from schedule $\sigma(v_j, e_2)$. In this case, since both token production times following from $\sigma(v_j, e_1)$ and token consumption times following from $\sigma(v_j, e_2)$ are delayed by Δ in the same firing, tokens arrive in time to enable schedule $\sigma(v_j)$. Since this holds for all pairs e_1 and e_2 this implies that Equation (14) is true. □

Lemma 3 determines on which edge of an actor the token transfer parameter gets selected the minimum value in Algorithm 1. This result is used in Lemma 4 to establish which of the schedules constructed per edge determines the schedule constructed for the actor.

Lemma 3 *Given Assumption 1, an actor v_i and a parameter $p \in P(v_i)$. Then Algorithm 1 selects $\bar{p} = \check{p}$ iff there is a simple directed path from v_i to v_τ that includes an edge from $E(p)$.*

Proof. $\Rightarrow$ We have that for parameter p, Algorithm 1 selects $\bar{p} = \check{p}$, if (1) there is an edge e_{ij} with $\pi(e_{ij}) = p$ and $d(v_i) > d(v_j)$ or (2) there is an edge e_{hi} with $\gamma(e_{hi}) = p$ and $d(v_i) > d(v_h)$. By construction of VRDF graphs, we have that the existence of an edge e_{hi}, with $\gamma(e_{hi}) = p$ and $d(v_i) > d(v_h)$, implies the existence of an edge e_{ih}, with $\pi(e_{ih}) = p$ and $d(v_i) > d(v_h)$. This means we can restrict our focus to case (1). We have that $d(v_i) > d(v_j)$ implies that there is a simple directed path from v_j to v_τ that does not include v_i. Appending this simple directed path to e_{ij} creates the required simple directed path.

$\Leftarrow$ If there is a simple directed path from v_i to v_τ that includes an edge from $E(p)$, then, by consistency, every simple directed path from v_i to v_τ includes an edge from $E(p)$. This implies the existence of an edge $e_{ij} \in E(p)$ to actor v_j with $d(v_j) < d(v_i)$, where v_j is allowed to equal v_τ. If there is such an edge, then Algorithm 1 selects $\bar{p} = \check{p}$. □

Lemma 4 *Given Assumption 1. Then the following holds*

$$\forall v_i \in V \setminus \{v_\tau\}\ \exists e_{ij} \in E(v_i) \bullet v_i \neq v_j \wedge d(v_j) \leq d(v_i) \wedge \sigma(v_i) = \sigma(v_i, e_{ij}) \quad (15)$$

Proof. For any parameter p of actor v_i, we can have (1) $\bar{p} = \hat{p}$, or (2) $\bar{p} = \check{p}$, or (3) $\bar{p} = \hat{p} = \check{p}$, i.e. the parameter is a constant.

If the token transfer parameter of v_i on edge e_1 falls into case (1), then $\sigma(v_i, e_1) \leq \bar{\sigma}(v_i, e_1)$. Else if the token transfer parameter of v_i on edge e_2 falls into case (2), then $\sigma(v_i, e_2) \geq \bar{\sigma}(v_i, e_2)$. Otherwise if the token transfer parameter of v_i on edge e_3 falls into case (3), then $\sigma(v_i, e_3) = \bar{\sigma}(v_i, e_3)$.

If there is an edge e_{ij} with $v_i \neq v_j \wedge d(v_j) \leq d(v_i)$, then this edge is on a simple directed path from v_i to v_τ. Given Assumption 1, Lemma 3 now tells us that parameter $p = \pi(e_{ij})$ has $\bar{p} = \check{p}$. This implies that e_{ij} falls into case (2) or (3). By consistency, we have that all parameters $q \in P(v_i) \setminus \{p\}$ are not on a simple directed path to v_τ. Which, by Lemma 3, means that $\bar{q} = \hat{q}$, which implies that parameters local to v_i and different from p fall into case (1) or (3).

This means that if there is an edge e_{ij} with $v_i \neq v_j \wedge d(v_j) \leq d(v_i)$, where $\pi(e_{ij}) = p$, then $\sigma(v_i, e_{ij}) \geq \bar{\sigma}(v_i, e_{ij})$ and for all edges $e \in E(v_i) \setminus E(p)$, $\sigma(v_i, e) \leq \bar{\sigma}(v_i, e)$.

By Theorem 1, we have that $\forall e, e' \in E(v_j) \bullet \bar{\sigma}(v_j, e) = \bar{\sigma}(v_j, e')$. Since $\sigma(v_i) = \max(\{\sigma(v_i, e) | e \in E(v_i)\})$, we have that if there is such an edge e_{ij}, with $v_i \neq v_j \wedge d(v_j) \leq d(v_i)$, then $\sigma(v_i)$ is determined by $\sigma(v_i, e_{ij})$. Since $v_i \neq v_\tau$ there is always such an edge e_{ij}. □

Lemma 5 states that if, under the constructed schedule for an actor, the schedule on an output edge remains within its bounds, then on this edge tokens arrive in time to enable the schedule of the token consuming actor constructed for this edge.

Lemma 5 *The following holds.*

$$\forall e_{ij} \in E \bullet \mathit{valid}(\sigma(v_i, e_{ij})) \implies \mathit{valid}(\sigma(v_j, e_{ij})) \quad (16)$$

Proof. According to Definition 9, we have that $\mathsf{valid}(\sigma(v_i, e_{ij}))$ is true iff $\mathsf{valid}(\sigma(v_i))$ and $\sigma(v_i, e_{ij}) = \sigma(v_i)$. This means that $\mathsf{valid}(\sigma(v_i, e_{ij}))$ is true if on all input edges of v_i tokens arrive before they are consumed, and that the constructed schedule of actor v_i is the constructed schedule of v_i on edge e_{ij} that can be linearly bounded. This implies that token production times on e_{ij} are conservatively bounded by the linear upper bound on production times. Furthermore token consumption times in schedule $\sigma(v_j, e_{ij})$ are conservatively bounded by the linear lower bound on consumption times. Since these bounds are taken into account when computing start times for schedules $\sigma(v_i, e_{ij})$ and $\sigma(v_j, e_{ij})$ in Algorithm 2, Equation (16) holds. □

In Theorem 2, the graph is traversed from actors that are furthest away from v_τ to actors that are closer to v_τ. For any actor we will show that if on all edges from actors that are closer to v_τ tokens arrive on time, i.e. Assumption 2 holds, and on all edges from actors further away from v_τ tokens arrive on time, then this actor produces its tokens on all edges to actors closer to v_τ on time. As we traverse the graph, Assumption 2 applies to ever fewer actors until it only applies to v_τ. The proof is concluded by showing that in fact this assumption holds for v_τ.

Assumption 2 *For actor v_j holds that on all edges e_{ij} with $d(v_i) \leq d(v_j)$ holds that $\mathit{valid}(\sigma(v_j, e_{ij}))$.*

Theorem 2 *Given Assumption 1, then the number of initial tokens as computed by Equation* (6) *is sufficient to let* v_τ *execute strictly periodically with period* τ.

Proof. The proof is by structural induction over the breadth-first tree as constructed in Algorithm 1.

Base step. Let actor v_i be a leaf of this tree, then given Assumption 2 it holds that $\sigma(v_i)$ is valid. This implies that, on any output edge e of actor v_i, we have $\mathsf{valid}(\sigma(v_i, e))$. This is because Lemma 4 states $\sigma(v_i) = \sigma(v_i, e)$.

Induction step. For any actor v_j, if on all edges e_{kj} from actors v_k, with $d(v_k) > d(v_j)$, it holds that $\mathsf{valid}(\sigma(v_j, e_{kj}))$, and on all edges e_{hj} from actors v_h, with $d(v_h) \leq d(v_j)$, it holds that $\mathsf{valid}(\sigma(v_h, e_{hj}))$, i.e. Assumption 2 holds, then $\mathsf{valid}(\sigma(v_j))$ holds. With $\mathsf{valid}(\sigma(v_j))$ true, we have, by Lemma 4, that on all edges e_{jl}, with $d(v_l) \leq d(v_j)$, $\mathsf{valid}(\sigma(v_j, e_{jl}))$.

Together with Lemma 5, this means that starting from the leaves of the breadth-first tree, we can traverse the tree in a breadth-first manner back to v_τ to reach the conclusion that $\mathsf{valid}(\sigma(v_\tau))$ given Assumption 2.

However, for actor v_τ Assumption 2 holds per construction. This is because there are no actors with a smaller than or equal distance, which implies that we only need to check edges from v_τ to itself. For each $p \in P(v_\tau)$, we have $\bar{p} = \hat{p}$. By Lemma 1, this implies that $\forall e \in E(v_\tau) \bullet \sigma(v_\tau, e) \leq \bar{\sigma}(v_\tau, e) = \sigma(v_\tau)$. Any edge $e_{\tau\tau}$ from v_τ to itself needs, by consistency, to have the same token production and consumption quantum. This implies that the schedule of the token producer, i.e. $\sigma(v_\tau, e_{\tau\tau})$ and the schedule of the token consumer, i.e. $\sigma(v_\tau, e_{\tau\tau})$, are delayed by the same delay Δ, which implies that tokens arrive in time. □

The next part of this section shows that Theorem 2 also holds if Assumption 1 is removed, i.e. if we allow parameters that are shared by two actors. We first show Lemma 6, which states that the schedule constructed per actor never has earlier start times than the schedules constructed for the parameter values $\bar{p}$. Together with the fact that for any shared parameter p, we have $\bar{p} = \hat{p}$, this will imply that the schedules constructed for the edges with token transfer parameter p will never determine the constructed schedule of the actor. In other words, there is always another edge for which the constructed schedule has later start times than the edge with the shared parameter.

Lemma 6 *The following holds*

$$\forall v_i \in V \; \forall e \in E(v_i) \bullet \bar{\sigma}(v_i, e) \leq \sigma(v_i) \tag{17}$$

Proof. For actor v_τ, we have by construction that for all edges $e \in E(v_\tau)$ that $\bar{\sigma}(v_\tau, e) = \sigma(v_\tau)$.

Every actor $v_i \neq v_\tau$ has an edge $\check{e}$ that is on a simple directed path to v_τ. Let the token transfer quantum of v_i on $\check{e}$ equal parameter p. There are three cases for p: (1) $\bar{p} = \hat{p}$, or (2) $\bar{p} = \check{p}$, or (3) $\bar{p} = \hat{p} = \check{p}$, i.e. p is constant.

By construction of VRDF graphs, we have that if p falls into case (1), then there has to be an edge $e_{ij} \in E(v_i) \backslash E(p)$ that is on a simple directed path to v_τ that transfers the value of p to another actor, and has constant token transfer quanta, i.e. the token production parameter of e_{ij} falls into case (3). Therefore, every actor different from v_τ always has a token production parameter that falls into case (2) or (3).

If p falls into case (2) or (3), then it follows from Lemma 1 that for all values of p, $\bar{\sigma}(v_i, \check{e}) \leq \sigma(v_i, \check{e})$. It follows from Equation (12) that $\forall e \in E(v_i) \bullet \sigma(v_i, e) \leq \sigma(v_i)$, which implies that $\bar{\sigma}(v_i, \check{e}) \leq \sigma(v_i, \check{e}) \leq \sigma(v_i)$.

Since Theorem 1 states that $\forall e, e' \in E(v_i) \bullet \bar{\sigma}(v_i, e) = \bar{\sigma}(v_i, e')$, it follows that $\forall e \in E(v_i) \bullet \bar{\sigma}(v_i, e) \leq \sigma(v_i)$. □

Lemma 7 shows that if an actor uses a shared parameter p and tokens arrive on time for the constructed schedule of this actor for parameter value $\bar{p}$, then tokens also arrive on time for the constructed schedule of this actor for any other value of p.

Lemma 7 *Given two actors* $v_a, v_b \in V$*, with* $v_a \neq v_b$ *and* $p \in P(v_a) \cap P(v_b) \neq \emptyset$*. Then if* $\mathsf{valid}(\sigma(v_a))$ *and* $\mathsf{valid}(\sigma(v_b))$ *for value* $\bar{p}$*, then* $\mathsf{valid}(\sigma(v_a))$ *and* $\mathsf{valid}(\sigma(v_b))$ *for every value of* p.

Proof. Let edge e_{ab} be the edge that transfers the values of parameter p, i.e. with $\varphi(e_{ab}) = p$. By construction of VRDF graphs there are no initial parameter values on this edge, i.e. $\delta(e_{ab}) = 0$. Furthermore, $\pi(e_{ab}) = \gamma(e_{ab}) = 1$, which implies that $q_a = q_b$ and $\omega(v_a) = \omega(v_b)$. Since there are no initial tokens on e_{ab}, the value of p used in firing $f \in \mathbb{N}^*$ of v_a equals the value of p used in firing f of v_b. Let there be edges e_{ax} and e_{yb}, with $\pi(e_{ax}) = p$ and $\gamma(e_{yb}) = p$. It follows from Lemma 1 that the difference between the start times of firings $f + 1$ and f in $\sigma(v_a, e_{ax})$ equals the difference between the start times of firings $f + 1$ and f in $\sigma(v_b, e_{yb})$.

By construction of VRDF graphs, we have that for every $p \in P(v_a) \cap P(v_b) \neq \emptyset$ we have that $\bar{p} = \hat{p}$. By Lemma 1, this implies that $\sigma(v_a, e_{ax}) \leq \bar{\sigma}(v_a, e_{ax})$ and $\sigma(v_b, e_{yb}) \leq \bar{\sigma}(v_b, e_{yb})$.

Since, by Lemma 6, $\bar{\sigma}(v_a, e_{ax}) \leq \sigma(v_a)$, a value of p smaller than $\bar{p} = \hat{p}$ leads to a difference between the actual start time of v_a and the start time according to $\sigma(v_a, e_{ax})$ that is increased by $\Delta \geq 0$. Similarly, since we have by Lemma 6 that $\bar{\sigma}(v_b, e_{yb}) \leq \sigma(v_b)$, a value of p that is smaller than $\bar{p}$ leads to a difference between the actual start time of v_b and the start time according to $\sigma(v_b, e_{yb})$ that is also increased by Δ. Since a VRDF graph has linear temporal behaviour a delay Δ in token production times on e_{ax} does not lead to a delay $\Delta' > \Delta$ in token arrival times on e_{yb}. Therefore, on edge e_{yb}, if tokens are consumed after they arrived for value $\bar{p}$, then for any value of p tokens are consumed after they arrived. □

The fact that the number of initial tokens as computed with Equation (6) is sufficient to let v_τ execute strictly periodically for any VRDF graph is shown by Theorem 3.

Theorem 3 *The number of initial tokens as computed by Equation* (6) *is sufficient to let v_τ execute strictly periodically with period τ.*

Proof. Theorem 2 states that this theorem is true given Assumption 1. Let v_a and v_b share a parameter p, i.e. $p \in P(v_a) \cap P(v_b) \neq \emptyset$. Then (1) there is a simple directed path from v_a to v_τ that does not include v_b, or (2) there is a simple directed path from v_b to v_τ that does not include v_a, or (3) $v_a = v_\tau$ or (4) $v_b = v_\tau$. If (1) or (2) is true then this implies that on that path there is no token transfer quantum that is parameterised in p. By Lemma 7, removal of Assumption 1 for p does not affect the validity of $\sigma(v_a)$ or $\sigma(v_b)$. This means that there is no schedule $\sigma(v_i)$ of an actor v_i on these paths that is affected by a change in value of p. Furthermore in case of (3) or (4), Lemma 7 tells us that for every value of p the schedule $\sigma(v_\tau)$ remains valid. Since v_a and v_b are an arbitrary pair of actors that share a parameter p, we can remove Assumption 1. □

8. Experimental Results

In this section we apply the presented algorithm to compute buffer capacities to the H.263 video decoder of which the task graph is shown in Figure 1. Furthermore, we use this application to compare our approach with the approach presented in [15].

In this task graph, we have a block reader (BR), a variable-length decoder (VLD), a dequantiser (DQ), an inverse discrete cosine transformation (IDCT), a motion compensator (MC), and a digital-to-analog converter (DAC) task. The BR task reads blocks of 2048 bytes from a compact-disc. The number of bytes, m, consumed per picture by the VLD is data-dependent. Further, also the number of blocks produced per picture, n, by the VLD task is data-dependent. In order to enable the MC task to construct a picture, the VLD communicates the number of blocks together with any motion vectors, $1[n]$, to the MC task. In this application, we require that the DAC task executes strictly periodically such that once every 33 ms a picture is displayed.

Let us assume that all input sequences have a resolution of 352x288, i.e. CIF format. Let us assume that the maximum number of bytes per picture is $\hat{m} = 6536$. For this resolution, a picture is divided into 396 macro blocks. Each macro-block consists of 6 blocks. This implies that the maximum number of blocks per picture is $\hat{n} = 6 * 396 = 2376$.

Worst-case response times of $\kappa(w_{BR}) = 10\,\text{ms}$, $\kappa(w_{VLD}) = 33\,\text{ms}$, $\kappa(w_{DQ}) = 14\,\mu\text{s}$, $\kappa(w_{IDCT}) = 14\,\mu\text{s}$, and $\kappa(w_{MC}) = 33\,\text{ms}$ enable satisfaction of the throughput constraint given sufficiently large buffers.

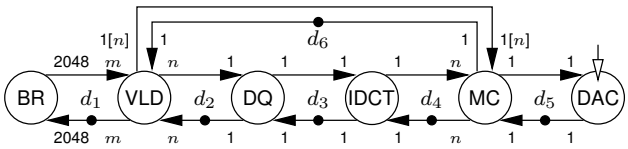

Figure 5. H.263 VRDF graph.

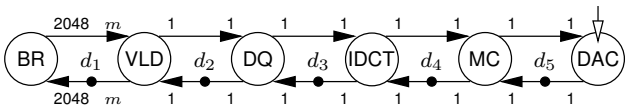

Figure 6. Alternative H.263 VRDF graph.

Figure 5 shows the VRDF graph that models the task graph from Figure 1. The presented algorithm results in a number of initial tokens of $d_1 = 17099$, $d_2 = 4734$, $d_3 = 2$, $d_4 = 4772$, $d_5 = 2$, and $d_6 = 4$. With our dataflow simulator, we have verified that these buffer capacities are indeed sufficient to let the DAC task execute strictly periodically with the required period. This was done by executing the dataflow graph and selecting subsequent parameter values in a uniform way from the specified intervals. The graph was executed for 10^6 executions of the DAC actor.

As presented in [15], an alternative approach to deal with a data-dependent number of tokens produced or consumed is to change the implementation from (1) a task graph that has variation in the number of containers that are transferred per execution and fixed sized containers to (2) a task graph that has no variation in the number of containers that are transferred per execution and a variable container size. In this alternative approach every container is extended with a value denoting the size of that container. However, this approach cannot be applied on the buffer between the BR and VLD tasks, because the BR task does not know what container size the VLD expects. This approach can be applied for the buffers between the VLD and the DQ task, between the DQ and the IDCT task, and between the IDCT and the MC task. Figure 6 shows the resulting VRDF graph. In these buffers, the container size that is used to compute buffer capacities is, in this alternative approach, the maximum container size and equals 2376 blocks.

For the VRDF graph of Figure 6, we have that a response time of 33 ms for the VLD, DQ, IDCT, and MC tasks allows for satisfaction of the throughput constraint. Except for the buffer between the BR and VLD task, the graph that is shown in Figure 6 is a single-rate dataflow graph. If we only consider this single-rate dataflow subgraph, then a sufficient number of initial tokens can be

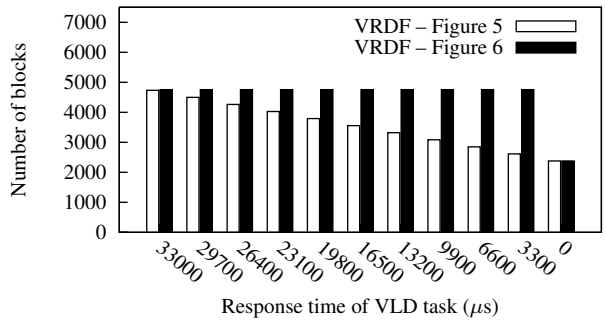

Figure 7. Required buffer capacity for buffer between VLD and DQ tasks.

determined using so-called maximum cycle mean analysis [14]. Application of maximum cycle mean analysis results in $d_2 = d_3 = d_4 = d_5 = 2$, while the number of initial tokens d_1 cannot be computed. For the buffers between VLD and DQ, between DQ and IDCT, and between IDCT and MC the container size in this alternative approach is 2376 blocks. This results in an increase from a buffer capacity of 2 blocks to a buffer capacity of 4752 blocks for the buffer between the DQ and IDCT tasks.

Another difference between the two approaches is the following. In Figure 7, the required number of initial tokens d_2 is plotted for various response times of the VLD task. What is clear from these results is that the smaller container size allows to reduce the buffer capacity in a more gradual manner.

Furthermore, the number of blocks in a picture can attain the value zero. In the approach with variable sized containers, the DQ and IDCT will, in this case, still execute and read the container size, while in our approach these tasks will not be enabled by blocks of this picture. Using the approach from [15] on the part of the task graph on which it can be applied, therefore does not have any advantages over the presented approach.

9. Conclusion

Applications such as audio and video decoders include tasks that produce and consume a data-dependent amount of data. We have presented a dataflow model that allows us to model this data-dependent communication behaviour, together with an algorithm that computes buffer capacities that guarantee satisfaction of a throughput constraint. The presented dataflow model allows for a straightforward check to determine whether any given graph is a valid input for our algorithm.

Important differences with current approaches are that current approaches either do not allow the communication behaviour to change in every execution or do not have a check that guarantees that bounded buffer capacities exist.

We expect that the presented dataflow model can be extended to allow actors to have a cyclic sequence of phases. This extension would allow us to model the variable-length decoding and motion compensation tasks of the H.263 video decoder in more detail, potentially resulting in smaller buffer capacities. The results of this paper provide a basis for a mapping flow that computes scheduler settings and buffer capacities such that end-to-end real-time requirements are satisfied for applications with data-dependent inter-task communication.

References

[1] B. Bhattacharya and S. S. Bhattacharyya. Consistency Analysis of Reconfigurable Dataflow Specifications. In *Proc. Int'l Workshop on System Architecture Modeling and Simulation*, 2001.

[2] B. Bhattacharya and S. S. Bhattacharyya. Parameterized Dataflow Modeling for DSP Systems. *IEEE Transactions on Signal Processing*, 49(10), 2001.

[3] J. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model.* PhD thesis, University of California at Berkeley, 1993.

[4] G. R. Gao *et al.* Well-behaved Dataflow Programs for DSP Computation. In *Proc. Int'l Conference on Acoustics, Speech, and Signal Processing*, 1992.

[5] A. Girault *et al.* Hierarchical Finite State Machines with Multiple Concurrency Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6), 1999.

[6] W. Haid and L. Thiele. Complex Task Activation Schemes in System Level Performance Analysis. In *Proc. CODES+ISSS*, 2007.

[7] M. Jersak *et al.* Performance Analysis of Complex Embedded Systems. *International Journal of Embedded Systems*, 1(1-2), 2005.

[8] E. A. Lee. Consistency in Dataflow Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 2(2), 1991.

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

[10] E. A. Lee and T. M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5), 1995.

[11] A. Maxiaguine *et al.* Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs. In *Proc. CODES+ISSS*, 2004.

[12] S. Neuendorffer and E. A. Lee. Hierarchical Reconfiguration of Dataflow Models. In *Proc. MEMOCODE*, 2004.

[13] M. Pankert *et al.* Dynamic Data Flow and Control Flow in High Level DSP Code Synthesis. In *Proc. Int'l Conference on Acoustics, Speech, and Signal Processing*, 1994.

[14] S. Sriram and S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker Inc., 2000.

[15] M. Sen *et al.* Modeling Image Processing Systems with Homogeneous Parameterized Dataflow Graphs. In *Proc. Int'l Conference on Acoustics, Speech, and Signal Processing*, 2005.

[16] S. Stuijk *et al.* Exploring Trade-Offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs. In *Proc. DAC*, 2006.

[17] P. Wauters *et al.* Cyclo-Dynamic Dataflow. In *Proc. Workshop on Parallel and Distributed Processing*, 1996.

[18] M. H. Wiggers *et al.* Efficient Computation of Buffer Capacities for Multi-Rate Real-Time Systems with Back-Pressure. In *Proc. CODES+ISSS*, 2006.

[19] M. H. Wiggers *et al.* Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *Proc. DAC*, June 2007.

[20] M. H. Wiggers *et al.* Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. In *Proc. RTAS*, 2007.