Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs

Maarten H. Wiggers
University of Twente
Enschede, The Netherlands
m.h.wiggers@utwente.nl

Marco J.G. Bekooij
NXP Semiconductors
Eindhoven, The Netherlands
marco.bekooij@nxp.com

Gerard J.M. Smit
University of Twente
Enschede, The Netherlands
g.j.m.smit@utwente.nl

ABSTRACT

A key step in the design of cyclo-static real-time systems is the determination of buffer capacities. In our multi-processor system, we apply back-pressure, which means that tasks wait for space in output buffers. Consequently buffer capacities affect the throughput. This requires the derivation of buffer capacities that both result in a satisfaction of the throughput constraint, and also satisfy the constraints on the maximum buffer capacities. Existing exact solutions suffer from the computational complexity that is associated with the required conversion from a cyclo-static dataflow graph to a single-rate dataflow graph. In this paper we present an algorithm, with polynomial computational complexity, that does not require this conversion and that obtains close to minimal buffer capacities. The algorithm is applied to an MP3 play-back application that is mapped on our multi-processor system. For this application, we see that a cyclo-static dataflow model can reduce the buffer capacities by 50% compared to a multi-rate dataflow model.

Categories and Subject Descriptors: C.3 [Special Purpose and application based systems]: Real-time and embedded systems

General Terms: Algorithms, Design, Performance

Keywords: System-on-Chip, Dataflow, Buffer Capacity

1. INTRODUCTION

Decreasing feature sizes have made it possible to implement multiple processing cores on a single chip, resulting in so-called Multi-Processor System-on-Chip (MPSoC) designs. These MPSoCs provide a high data processing throughput in a cost and energy-efficient way, making them an ideal match with multi-media applications as can be found in TV-sets, set-top boxes, and smartphones.

MPSoCs operate on multiple streams of data that often have temporal constraints, such as throughput and latency. These streams have firm or soft real-time constraints. In the mentioned application domain, throughput constraints typically dominate over latency constraints.

For firm real-time streams we want to guarantee that no deadline is missed, because this would result in a severe quality degradation. Therefore, we model the processing performed on these streams with Cyclo-Static Dataflow (CSDF) graphs [3, 8], of which we can analytically derive the cycle that determines the throughput after conversion to a Single-Rate Dataflow (SRDF) graph [3]. Tasks are modelled by the vertices of a CSDF graph, which are called actors.

As discussed and shown in [3] and [8], CSDF graphs are more expressive than Multi-Rate Dataflow (MRDF) graphs [6]. This means that both a larger class of applications can be modelled and that more detailed dependencies can be included, which often leads to lower resource requirements.

An essential step when programming a multi-processor system is the determination of buffer capacities. In our multi-processor system, tasks start their execution on an assigned processor based on the availability of containers that signal the presence of data or space in a first-in first-out (FIFO) buffer with a fixed capacity. The fact that the start of a task execution is dependent on the availability of space results in so-called back-pressure. Therefore, in our system, the buffer capacity has an influence on the start times of tasks and consequently on the throughput. Applying back-pressure has the advantage that the system does not require means to control jitter, such as e.g. traffic shapers, in order to prevent buffer overflow.

However, determining whether particular buffer capacities allow a throughput that satisfies the constraint is a complex task. In order to find exact results, first a conversion from a cyclo-static to a single-rate dataflow graph is required, which can result in an exponential number of vertices [9], after which the throughput of each cycle in the SRDF graph can be determined [4]. Algorithms that have a polynomial complexity for SRDF graphs, therefore have an exponential complexity for CSDF graphs.

Contribution. The contribution of this paper is an algorithm based on a min-cost network flow formulation that obtains close to minimal buffer capacities for CSDF graphs that satisfy both the temporal constraint as well as any buffer capacity constraints that are for instance caused by finite memory sizes. The presented algorithm has a polynomial complexity for CSDF graphs, and results in small run-times and memory requirements.

This is accomplished as follows. In our multi-processor system [1], we only use pre-emptive schedulers that provide resource budget guarantees [7]. These schedulers allow the derivation of the response time of a task based on only the execution time of the task and the scheduler settings. This response time is therefore independent of the behaviour of other tasks. Using these response times, a conservative schedule of actor executions is constructed that meets the throughput constraint. From this schedule we derive sufficient buffer capacities for the FIFO buffers. This is possible, because our system has monotonic temporal behaviour, as explained in Section 2, which guarantees that the schedule as determined at run-time will not lead to later container production times.

Related work. The presented work is a generalisation of [12], in which an algorithm is presented that obtains close to minimal buffer capacities given buffer capacity and throughput constraints for a class of MRDF graphs. In this paper, we (1) present an algorithm

DAC 2007, June 4–8, 2007, San Diego, California, USA.

for a more expressive dataflow model, and (2) remove the constraint on the topology of the dataflow graph.

Two alternative approaches that determine buffer capacities for CSDF graphs are known to us. The back-tracking approach to derive a schedule as applied by [3] can be extended to deal with temporal constraints, constraints on buffer capacities, and buffer capacity minimisation. Another approach is to back-track over the possible buffer capacities of the CSDF graph, and check whether the temporal constraints are satisfied by applying MCM analysis on the corresponding SRDF graph [10]. Both approaches suffer from an exponential space and time complexity, because they schedule SRDF actors, but are able to satisfy the throughput requirement and buffer capacity constraints for a larger set of problem instances, because they are exact.

The organization of this paper is as follows, relevant properties of CSDF graphs are summarised in Section 2. Section 3 provides the basic ideas behind the approach pursued in this paper. In Section 4 we determine a schedule of actor phases. These schedules are used in Section 5 to determine a minimum distance between two actors connected by an edge. These distances form the constraint set of the network flow problem that determines start times to minimise the buffer capacity as presented in Section 6. By applying the algorithm on an MP3 playback case study, we investigate the run-time and accuracy of the algorithm in Section 7.

2. ANALYSIS MODEL

The input to our mapping flow is a CSDF [3] graph that models the application. A CSDF graph is a directed graph $G = (V, E, \delta, \rho, \pi, \gamma, \theta)$ that consists of a finite set of actors V, and a set of directed edges, $E = \{(v_i, v_j) | v_i, v_j \in V\}$. Actors synchronise by communicating tokens over edges that represent queues. The graph G has an initial token placement $\delta : E \rightarrow \mathbb{N}$. An actor v_i has $\theta(v_i)$ distinct phases of execution, with $\theta : V \rightarrow \mathbb{N}$, and transitions from phase to phase in a cyclic fashion. An actor is enabled to fire when the number of tokens that will be consumed is available on all its input edges. The number of tokens consumed in firing k by actor v_i is determined by the edge and the current phase of the token consuming actor, $\gamma : E \times \mathbb{N} \rightarrow \mathbb{N}$, and therefore equals $\gamma(e, ((k-1) \mod \theta(v_i)) + 1)$ tokens. The specified number of tokens is consumed in an atomic action from all input edges when the actor is started. The response time $\rho(v_i, f)$, $\rho : V \times \mathbb{N} \rightarrow \mathbb{R}$, is the difference between the finish and the start time of phase f of actor v_i. The response time of actor v_i in firing k is therefore $\rho(v_i, ((k-1) \mod \theta(v_i)) + 1)$. To ease the notation we introduce $\rho_f : V \times \mathbb{N} \rightarrow \mathbb{R}$ that provides the response time of a firing, i.e. $\rho_f(v_i, k) = \rho(v_i, ((k-1) \mod \theta(v_i)) + 1)$. When actor v_i finishes, then it produces the specified number of tokens on each output edge $e = (v_i, v_j)$ in one atomic action. The number of tokens produced in a phase will be denoted by $\pi : E \times \mathbb{N} \rightarrow \mathbb{N}$.

For edge $e = (v_i, v_j)$, we define $\Pi(e) = \sum_{f=1}^{\theta(v_i)} \pi(e, f)$ as the number of tokens produced in one cyclo-static period, and $\Gamma(e) = \sum_{f=1}^{\theta(v_j)} \gamma(e, f)$ as the number of tokens consumed in one cyclo-static period. We further define the topology matrix Ψ as an $|E| \times |V|$ matrix, where

$$\Psi_{mi} = \begin{cases} \Pi(e_m) & \text{if } e_m = (v_i, v_j) \text{ and } v_i \neq v_j \\ -\Gamma(e_m) & \text{if } e_m = (v_j, v_i) \text{ and } v_i \neq v_j \\ \Pi(e_m) - \Gamma(e_m) & \text{if } e_m = (v_i, v_i) \\ 0 & \text{otherwise} \end{cases}$$

If the rank of Ψ is $|V| - 1$, then a connected CSDF graph is said to be consistent [3]. A consistent CSDF graph requires queues with finite capacity, while an inconsistent CSDF graph requires infinite queue capacity.

We define the vector $\mathbf{s}$ of length $|V|$, for which holds $\Psi \mathbf{s} = \mathbf{0}$, and which determines the relative firing frequencies of the cyclo-static periods. The repetition vector $\mathbf{q}$ of the CSDF graph determines the relative firing frequencies of the actors and is given by

$$\mathbf{q} = \Theta \mathbf{s} \quad \text{with} \quad \Theta_{ik} = \begin{cases} \theta(v_i) & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

The repetition rate $\mathbf{q}_i$ of actor v_i is therefore the number of phases of v_i within one cyclo-static period times the relative firing frequency of the cyclo-static period.

In order to focus on the main concepts, we assume that every actor v_i implicitly has a self-edge (v_i, v_i) with a single initial token and that this actor produces and consumes a token from this self-edge in every firing.

For a strongly connected and consistent CSDF graph, we can specify a required period μ within which on average every actor v_i should fire $\mathbf{q}_i$ times. The throughput of the graph relates to μ^{-1}. In the remainder of this paper, we assume that the required period μ is given.

2.1 Monotonic Execution

If a CSDF graph is executed in a self-timed manner, then actors start execution as soon as they are enabled. If each actor either has a constant response time, or has a self-cycle with one initial token, then the graph maintains a FIFO ordering of tokens, since queues by definition maintain a FIFO ordering of tokens.

An important property is that self-timed execution of a strongly connected CSDF graph that maintains a FIFO ordering of tokens is monotonic in time, which is defined as follows.

DEFINITION 1. *A CSDF graph executes monotonically in time if no decrease in response time or start time of any firing k of any actor v_i can lead to a later enabling of firing l of actor v_j.*

If a CSDF graph G maintains FIFO ordering of tokens, then the self-timed execution of G is monotonic. This is because a decrease in response time or start time can only lead to earlier token production times, and therefore only to an earlier actor enabling.

2.2 Examples

An example CSDF graph is shown in Figure 1 in which data flows from actor v_p to actor v_c, where v_p has the response time sequence $r_p = \langle 2, 1 \rangle$ and v_c has the response time sequence $r_c = \langle 1, 1 \rangle$. The vector $\mathbf{s}$ is $[2\,3]^{\mathrm{T}}$, and the repetition vector is $\mathbf{q} = [4\,6]^{\mathrm{T}}$. One instance of the problem discussed in this paper is to find a token placement δ such that the period μ of the graph in Figure 1 is 6. The presented algorithm will tell us that the buffer capacity $d_2 = 3$ is sufficient to satisfy the constraints.

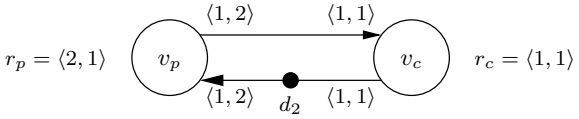

Figure 1: Example buffer capacity problem.

We will show that CSDF can model a larger class of tasks than MRDF. Figure 2 shows an example CSDF graph taken from [8] in which merging the actors v_1 and v_3 into an MRDF actor v_m will lead to deadlock, i.e. this creates a cycle with no initial tokens where v_2 requires a token on an edge from the actor v_m and the actor v_m requires a token from actor v_2 in order to be enabled. A CSDF actor can merge the actors v_1 and v_3 without introducing deadlock, since a CSDF actor can still allow the firing sequence $\langle v_1, v_2, v_3 \rangle$. Therefore, the task consisting of the code segments as

modelled by actors v_1 and v_3 can be modelled with a CSDF actor, but should not be modelled with an MRDF actor. This shows that CSDF is more expressive than MRDF.

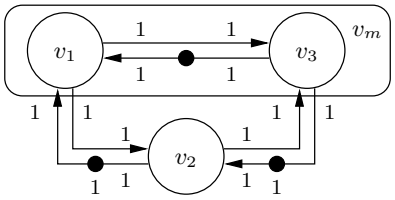

Figure 2: Deadlock if v_m is an MRDF actor.

2.3 Analysis model and implementation

We have established that CSDF graphs execute monotonically in time. As explained in [12], we construct the CSDF graph in such a way that there is a one-to-one correspondence between the task graph and the CSDF graph, with the only important difference that tasks can have a smaller response time than actors and produce their containers before they finish, while actors produce the corresponding tokens when they finish. Therefore we arrive at the conclusion that tasks produce their containers no later than the corresponding actors produce their tokens.

In the next section the basic idea is presented for the algorithm that will derive buffer capacities that satisfy the throughput and buffer capacity constraints.

3. BASIC IDEA

The basic idea of the algorithm is as follows. First a schedule of phase start times is constructed for each CSDF actor that satisfies the throughput constraint, i.e. actor v_i fires $\mathbf{q}_i$ times within μ. The schedule of phase start times is constructed independent of other actors in the graph, with the objective to minimise the difference between linear bounds on the token consumption and production.

On each edge, e.g. edge (v_p, v_c) in Figure 1, we have that tokens cannot be consumed before they are produced. The bounds on token consumption and production enable us to derive for each edge a minimal distance β between the start times of the first phases, e.g. between the first phases of v_c and v_p. These minimum distances form a set of constraints and using a min-cost network flow formulation we will determine the start times of the first phases such that all these constraints are satisfied and the sum of the distances between the start times is minimised. The topology of the graph can necessitate distances between the start times of the first phases that are larger than the previously determined distance β.

When on each edge the distance between the start times of the first phases is determined, the bounds on the token consumption and production enable us to derive sufficient buffer capacities to sustain the constructed schedules of phase start times.

Note that while buffer capacities are derived such that the constructed schedule always has sufficient tokens available, tasks in the implementation do experience back-pressure. This is because tasks execute in a self-timed fashion, which means that tasks start as soon as they are enabled, where a task is enabled if sufficient full containers are present on all input FIFO buffers and sufficient empty containers are present on all output FIFO buffers. And furthermore, since (1) start times in the constructed schedule are only delayed compared to the self-timed schedule, and (2) the implementation has monotonic temporal behaviour, the self-timed schedule will also meet the throughput constraint.

Figure 3 illustrates the derivation of the required number of initial tokens on the edge $b = (v_c, v_p)$ of the CSDF graph as shown in Figure 1. The schedule of phase start times of actor v_p results in token consumption and production times. The token consumptions of v_p are bounded by c_u^b, and the token productions by v_p on edge $e = (v_p, v_c)$ are bounded by p_l^e. Similarly for v_c, the token consumptions from edge e are bounded by c_u^e and the productions on edge b are bounded by p_l^b. Since no token can be consumed prior to production we have that $p_l^e \geq c_u^e$, which results in a minimum distance β between the start times of the first phases of v_c and v_p. The required number of tokens on (v_c, v_p) is the maximum difference between the number of tokens that are consumed by v_p and the number of tokens produced by v_c.

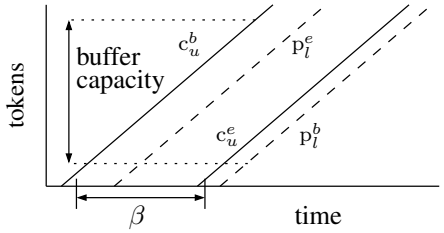

Figure 3: In order to minimise the buffer capacity, the distances $c_u^b - p_l^e$, $p_l^e - c_u^e$, and $c_u^e - p_l^b$ are minimised.

4. ACTOR SCHEDULE

In this section a schedule of phase start times is constructed, such that the throughput constraint and the constraint formed by the number of tokens on the self-edge are satisfied, and the difference between the linear upper and lower bounds is minimised, since minimising this difference contributes to minimisation of the buffer capacities.

This is accomplished by first constructing a free running schedule of the phases, then creating a linear lower bound on this free running schedule, and subsequently delaying the start times of the phases to minimise the difference between the linear upper and lower bounds.

4.1 Free running schedule

For each actor v_i, we first construct a free running schedule, which is the self-timed schedule that results after removal of all edges $(v_i, v_j), v_i \neq v_j$ and $(v_j, v_i), v_j \neq v_i$. The free running schedule of v_i is therefore the schedule that results when v_i starts as soon as v_i is enabled by tokens on its self-edge. The start time of firing k of actor v_i in the free running schedule is given by Equation (1).

$$s(v_i, k) = \begin{cases} s(v_i, k-1) + \rho_f(v_i, k-1) & \text{if } k > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Figure 4 shows the production time of each token that is produced in $\mathbf{s}_i\theta(v_p)$ phases of the free running schedule of actor v_p from Figure 1.

4.2 Linear lower bound

In this section we will construct a linear lower bound, $p_l^e(t) = \alpha_e t + \beta_e^p$, on the number of tokens produced by actor v_i on edge $e = (v_i, v_j) \in E$ by first deriving the slope α_e and then the offset β_e^p.

In Section 4.3 we will construct a schedule such that $\mathbf{s}_i\theta(v_i)$ firings occur in μ time, therefore $\mathbf{s}_i\Pi(e)$ tokens will be produced by actor v_i on edge e in μ time. The slope of the linear lower bound on the token production on edge e is therefore $\alpha_e = {}^{\mathbf{s}_i\Pi(e)}/_{\mu}$.

The linear lower bound needs to be conservative to the free running schedule. We obtain more accurate results by observing that

the lower bound needs not be conservative for every token that is produced, but only for a set of relevant token productions. In Section 4.3 we will create a schedule in which the last phase finishes at $\mu/\mathbf{s}_i$. The bound should therefore also be conservative to the smallest relevant token production that occurs in the last phase when the last phase finishes at $\mu/\mathbf{s}_i$.

We will now construct the set of relevant token productions. This is based on the observation that the lower bound only needs to be conservative for a cumulative number of tokens that enables a firing of actor v_j. The bound therefore needs to be conservative when token $\sum_{f=1}^{g} \gamma(v_j, f)$ is produced, with $g \geq 1, g \in \mathbb{N}$. Since the production rates are periodic with period $\theta(v_i)$ and the consumption rates are periodic with $\theta(v_j)$ the bound is conservative for all g if the bound is conservative for $1 \leq g \leq \text{lcm}(\theta(v_i), \theta(v_j))$, where lcm stands for the least common multiple.

This can result in problematic run-times since the result of a least common multiple can be much larger than its operands. We therefore apply the following approach, for which we first define λ_e as the greatest common divisor (gcd) of the total production rate and the individual consumption rates on edge e, $\lambda_e = \text{gcd}(\{\Pi(e)\} \cup \{\gamma(e,g) | g \in [1, \theta(v_c)] \wedge \gamma(e,g) \neq 0\})$.

Starting from the first period of the free running schedule, the production times of every λ_e tokens will be periodic with $\theta(v_i)$ firings. This is because in $\theta(v_i)$ phases $\Pi(e)$ tokens are produced on edge e, and λ_e divides $\Pi(e)$. Furthermore, we have that every cumulative number of tokens consumed can be described as $a\Pi(e) + b\lambda_e$, with $a, b \in \mathbb{N}$. This leads to the conclusion that the lower bound leads to conservative enabling times of v_j on edge e if the bound is conservative for every λ_e tokens that are produced within the first period of the free running schedule of v_i.

In Figure 4 the lower bound on the free running schedule of actor v_p from Figure 1 is shown. In this example, the dominant constraint is the constraint that the bound needs to be conservative to the smallest multiple of λ_e tokens produced in phase $\theta(v_p)$, which is token 2 produced in phase $\theta(v_p) = 2$. By construction, phase $\theta(v_p)$ will finish in the schedule as constructed in Section 4.3 at $\mu/\mathbf{s}_p = 6/2 = 3$.

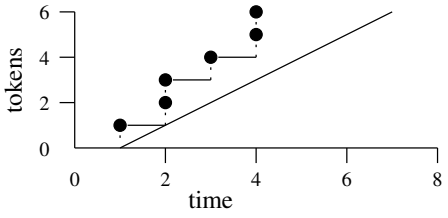

Figure 4: The linear lower bound on token productions of a free running schedule.

4.3 Start time postponement

Now that we have constructed the linear lower bound on the number of tokens produced, $p_l^e(t)$, we will delay the start times of the free running schedule. The objective of this start time postponement is to minimise the difference between the linear upper bound on the number of tokens consumed by v_i from edge $b = (v_j, v_i) \in E$, c_u^b, and the linear lower bound on the number of tokens produced by v_i, p_l^e, since this contributes to the minimisation of the buffer capacity as illustrated in Figure 3. This objective is achieved by minimising the difference between the time at which a relevant token m is produced in the constructed schedule and the time at which token m is produced according to the linear bound.

This schedule is constructed by letting phase $n\theta(v_i)$, with $n > 0, n \in \mathbb{N}$, finish at $n(\mu/\mathbf{s}_i)$ and by using these finish times as reference points to derive the other finish times. This results in a schedule that is periodic with a period of $\theta(v_i)$ firings, and we will therefore only derive the finish times during the first period.

When deriving this schedule there are two types of constraints. The first type of constraint is that we need to ensure that only one firing executes at any point in time. The second type of constraint is that the production time of any relevant token m according to the constructed schedule is not later than the production time of this relevant token m according to the linear bound $p_l^e(t)$.

The first type of constraint requires that the finish time $h(v_i, k)$ of firing k of actor v_i is smaller than or equal to $g(v_i, k+1)$, which is the start time of firing $k+1$ of actor v_i, as derived in Equation (2).

$$g(v_i, k) = \begin{cases} h(v_i, k) - \rho_f(v_i, k) & \text{if } k < \theta(v_i) \\ \frac{\mu}{\mathbf{s}_i} - \rho_f(v_i, k) & \text{otherwise} \end{cases} \quad (2)$$

The second type of constraint requires that the finish time $h(v_i, k)$ should be smaller than or equal to the upper bound on the production time of the smallest relevant token, m_e^k, that is not yet produced on edge e in firings 1 to $k-1$, as derived in Equation (3). The smallest relevant token is sufficient because every phase produces tokens in an atomic action.

$$m_e^k = \min(\{m | m = p\lambda_e > \sum_{u=1}^{k-1} \pi(e,u), p \geq 0, p \in \mathbb{N}\}) \quad (3)$$

According to the bound $p_l^e(t) = \alpha_e t + \beta_e^p$, as derived in Section 4.2, the number of tokens produced at time t is at least $p_l^e(t)$. Therefore for token $p_l^e(t) = m_e^k$ we have an upper bound on the production time that equals $m_e^k - \beta_e^p / \alpha_e$.

Since the finish time $h(v_i, k)$ should satisfy both types of constraints we obtain Equation (4) for firings k, with $1 \leq k < \theta(v_i)$. All finish times are now determined, because the finish time of firing $k = \theta(v_i)$ equals $\mu/\mathbf{s}_i$ by construction and we have that this schedule repeats with a period of $\theta(v_i$ firings.

$$h(v_i, k) = \min(\{g(v_i, k+1)\} \cup \{\frac{m_e^k - b_e}{\alpha_e} | e = (v_i, v_j)\}) \quad (4)$$

The start times as provided by Equation (2) determine the token consumption times on edge $b = (v_j, v_i) \in E$, which can be linearly bounded by $c_u^b(t) = \alpha_b t + \beta_b^c$.

Figure 5 shows the derived production times for the first $\mathbf{s}_p\theta(v_p)$ firings of actor v_p from Figure 1, by comparing the production times in Figure 4 and Figure 5 we see that finish times and thereby start times are delayed to construct a schedule that exactly satisfies the throughput constraint.

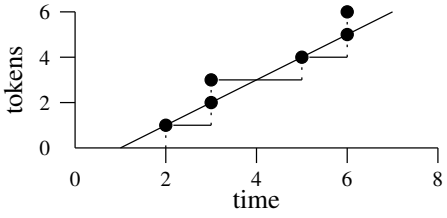

Figure 5: The linear lower bound on token productions and the derived finish times.

5. SCHEDULING CONSTRAINTS

In the previous section we have constructed actor schedules that satisfy the throughput constraint and minimise the difference between the linear upper bound on the number of tokens consumed and the linear lower bound on the number of tokens produced. In

this section we will determine a minimal distance β_{ij} between the start times of the first phases of the actors v_i and v_j that are connected by an edge $e = (v_i, v_j) \in E$ such that no token is consumed by v_j which is not already produced by v_i. This leads to the requirement that the number of tokens produced should be larger than or equal to the number of tokens consumed, and therefore $p_l^e \geq c_u^e$. Since the actor schedules are constructed such that v_i produces $\mathbf{s}_i\Pi(e)$ in μ time and v_j consumes $\mathbf{s}_j\Gamma(e)$ in μ time, we have by definition of $\mathbf{s}$ that the number of tokens produced in μ equals the number of tokens consumed in μ. This means that the slope of the linear bounds is equal, which leads to the conclusion that the minimum distance β_{ij} is such that $p_i^l = c_j^u$.

For the actor schedules as derived in the previous section, we have that, the lower bound on the number of tokens produced by v_i on edge e, $p_l^e(t) = 0$ at $t_p = {}^{-\beta_e^p}/_{\alpha_e}$. Furthermore we have that, the lower bound on the number of tokens consumed by v_j from edge e, $c_u^e(t) = 0$ at $t_c = {}^{-\beta_e^c}/_{\alpha_e}$. If we define $\tau_e^p = {}^{-\beta_e^p}/_{\alpha_e} - s(v_i, 1)$, and $\tau_e^c = s(v_j, 1) - {}^{-\beta_e^c}/_{\alpha_e}$, then we can derive the minimal distance between $s(v_j, 1)$ and $s(v_i, 1)$.

THEOREM 1. *If we let the first firing of actor v_j start τ_e later than the first firing of actor v_i, with $\tau_e = \tau_e^p + \tau_e^c - \lambda \left\lfloor {}^{\delta(e)}/_{\lambda_e} \right\rfloor \alpha_e$, then no token will be consumed before it is available.*

PROOF. If edge e does not have initial tokens, then the minimum distance τ_e is such that $p_i^l = c_j^u$. This implies that c_j^u should equal 0 at the same time as p_i^l equals 0 which is at $t_p = {}^{-\beta_e^p}/_{\alpha_e}$. The difference between t_p and $s(v_i, 1)$ is by definition τ_e^p, and the difference between $s(v_j, 1)$ and t_0 is again by definition τ_e^c.

With $\delta(e)$ initial tokens, we know that $\lambda \lfloor {}^{\delta(e)}/_{\lambda} \rfloor$ tokens contribute to an earlier enabling of v_j. According to the upper bound on the number of tokens consumed, actor v_j consumes a token every α_e time. Which means that, with $\delta(e)$ initial tokens, the schedule of actor v_j can start $\lambda \lfloor {}^{\delta(e)}/_{\lambda} \rfloor \alpha_c$ earlier. □

6. NETWORK FLOW PROBLEM

In this section we will determine for all actors a start time of the first phase, such that all constraints on the minimal differences between the start times of the first phases as derived in the previous section are satisfied and the buffer capacities are minimised. Because we have linearly bounded the actor schedules, minimising the buffer capacities involves minimising the differences between the start times of the first phases. This is achieved by adding a vertex v_0 to obtain $V_0 = V \cup \{v_0\}$, and adding edges (v_0, v_i) with $\beta_{0i} = 0$ to every actor $v_i \in V$ in the original graph to obtain a set of edges E_0. By minimising the difference in start times relative to the start time of v_0 the buffer capacities are minimised.

The problem formulation is shown in Algorithm 1. This problem is an instance of the dual to the uncapacitated minimum-cost network flow problem, which in our case can be solved in $O(|V|^4)$ time [2], with $|V|$ the number of CSDF actors.

Algorithm 1 Network Flow Problem

$$\min \sum_{v_i \in V} s(v_i, 1)$$

subject to

$$s(v_i, 1) - s(v_j, 1) \leq -\beta_{(v_i, v_j)} \quad \forall (v_i, v_j) \in E_0$$
$$s(v_0, 1) = 0$$

Now that the start time of the first phase of each actor has been determined, the required number of initial tokens on each cycle in the CSDF graph can be derived. Suppose that, on a cycle in the graph, edge $b = (v_c, v_p) \in E$ is the edge on which initial tokens are placed, then in Section 4 we have determined linear upper and lower bounds on the number of tokens consumed, $c_u^b(t)$, and produced, $p_l^b(t)$. While in Section 5 we have derived expressions that relate these bounds to the start time of the first phase of each actor. Now that the start times of the first phase are determined using the min-cost network flow formulation in Section 6, we can derive the ceiled difference between the upper bound on the number of tokens consumed and the lower bound on the number of tokens produced, $\lceil c_u^b(t) - p_l^b(t) \rceil$, which equals the required number of tokens to sustain the constructed schedule.

This results in a polynomial overall complexity. The derivation of the linear lower bound on token productions on an edge as well as the derivation of the finish times of the phases are both linear in the number of phases, which results in a complexity $O(|V||E|T)$, with $T = \max_i(\theta(v_i))$. The derivation of the minimum distances between the start times of the first phases is $O(|E|)$. Combining this with the complexity of the algorithm to solve the network flow problem results in an overall complexity of $O(|V|^4 + |V||E|T)$.

7. EXPERIMENTAL RESULTS

In this section we apply our algorithm to determine a minimal sum of buffer capacities for an MP3 playback application. The implementation of this application is first modelled as an MRDF graph, and subsequently as a CSDF graph. For both models the run-time and accuracy of the presented algorithm is compared with alternative approaches. The algorithm can be applied on the MRDF model, because MRDF is a subclass of CSDF, where MRDF restricts each actor to only have a single phase.

In the MP3 playback application, of which the MRDF graph is shown in Figure 6, with $R = 1152$, the compressed audio is decoded by the MP3 task into a 48 kHz audio sample stream. These samples are converted by the Sample Rate Converter (SRC) task into a 44.1 kHz stream, after which the Audio Post-Processing (APP) task enhances the perceived quality of the audio stream and sends the samples to a Digital to Analogue Converter (DAC).

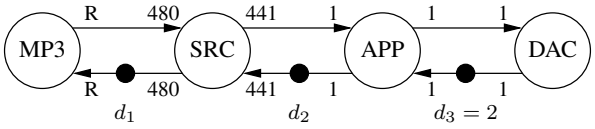

Figure 6: Dataflow model of the MP3 playback application

The repetition vector of the MRDF graph is $[5\ 12\ 5292\ 5292]^{\mathrm{T}}$. The frequency of the DAC is 44.1 kHz, leading to $r_{DAC} = {}^1/_{44100}$ s. The required period μ is fixed to $q_{DAC} \cdot r_{DAC} = 120$ ms to derive buffer capacities that guarantee a periodic execution of the DAC. The other response times are $r_{MP3} = 7.51$ ms, $r_{SRC} = {}^{\mu}/_{q_{SRC}} = {}^{120}/_{12} = 10$ ms, and $r_{APP} = {}^{\mu}/_{q_SRC} = {}^1/_{44100}$ s. In this experiment, the response time of the sample rate converter is varied to show the behaviour of the presented algorithm.

Table 1 lists the resulting buffer capacities for this experiment, both as determined by the algorithm presented in Section 6 and as determined through back-tracking where the throughput of the MRDF graph is determined with Maximum Cycle Mean (MCM) analysis [10], which is applied on the SRDF graph.

The behaviour of the MP3 task can however be modelled in more detail, which, as the following experiment shows, results in smaller buffer capacities. The decoding of every MP3 frame involves the decoding of the header, decoding of 2 granules, and decoding of 18 sub-bands that each consist of 32 samples per granule. Only after the sub-bands of a granule are decoded is the decoded data sent to

	Buffer Capacity						rel. diff. (%)
	d_1		d_2		$d_1 + d_2$		
	alg.	opt.	alg.	opt.	alg.	opt.	
r_{SRC}	2304	2016	882	882	3186	2898	10
$3/4 \cdot r_{SRC}$	2208	1824	772	988	2980	2812	6
$1/2 \cdot r_{SRC}$	2112	1536	662	772	2774	2308	20
$1/4 \cdot r_{SRC}$	2016	1536	552	551	2568	2087	23

Table 1: Our results (alg.), the optimal results (opt.) for the MRDF model of the MP3 playback application.

the SRC. This behaviour can be concisely modelled with a CSDF actor. In this case $R = \langle 0, 0, 18\text{x}32, 0, 18\text{x}32 \rangle$ in Figure 6.

The CSDF model has a different R, which results in a repetition vector $[195 \quad 12 \quad 5292 \quad 5292]^{\mathrm{T}}$, and further $r_{MP3} = \langle 670, 2700, 18\text{x}40, 2700, 18\text{x}40 \rangle$ µs. Again the response time of the sample rate converter is varied to show the behaviour of the presented algorithm.

Table 2 lists the resulting buffer capacities for this experiment, both as determined by the algorithm presented in Section 6 and as determined through back-tracking where the throughput of the CSDF graph is determined with Maximum Cycle Mean (MCM) analysis [10], which is applied on the SRDF graph. In this experiment, the total buffer capacity is reduced by at least 49% when using a CSDF model instead of an MRDF model.

	Buffer Capacity						rel. diff. (%)
	d_1		d_2		$d_1 + d_2$		
	alg.	opt.	alg.	opt.	alg.	opt.	
r_{SRC}	1056	960	882	882	1938	1842	5
$3/4 \cdot r_{SRC}$	928	576	772	910	1700	1486	14
$1/2 \cdot r_{SRC}$	800	480	662	661	1462	1141	28
$1/4 \cdot r_{SRC}$	672	480	552	551	1224	1031	19

Table 2: Our results (alg.), the optimal results (opt.) for the CSDF model of the MP3 playback application.

As shown in Tables 1 and 2, the accuracy of the algorithm decreases as the slack time and thereby the scheduling freedom increases. This is because the slack time is distributed over different firings of an actor in a locally optimal manner. The run-time of the algorithm for both the MRDF and the CSDF graph is in the order of 10^{-2} s.

We expect that many task to processor assignments and scheduler settings, which each lead to different response times, need to be evaluated in order to determine a configuration of our multiprocessor system that satisfies the temporal constraints. The low run-time of a single iteration enables this approach.

Even though we have been able to apply back-tracking in combination with MCM analysis for this example, we feel that, in general, this is not a feasible approach since one iteration of the Howard algorithm [4], which we used to derive the MCM, requires a minute.

Govindarajan's [5] linear programming formulation can be used to determine minimal buffer capacities for the MRDF model of the MP3 playback implementation. However, application of this approach on the MP3 playback application was infeasible, because the solver from the GNU Linear Programming Kit (GLPK) runs out of memory. Removal of the APP task from the graph enables the application of Govindarajan's approach, but still has a run-time of half an hour. Since CSDF is more expressive than MRDF, we expect that extensions of Govindarajan's approach to include CSDF graphs will also have problematic run-times and memory requirements.

It seems possible to extend the work presented in [11] to make it applicable to CSDF graphs. This approach does not require the conversion to the corresponding SRDF graph in order to determine the throughput, and leverages the fact that a strongly connected and consistent MRDF graph enters a periodic regime after a transitional phase. Even though good experimental results are provided, no bound on the complexity is provided, and we know of no bound on the length of the transitional phase. Furthermore, since the throughput is determined for every possible combination of buffer capacities, and we have that this number of combinations grows exponentially with the number of buffers, we expect that for graphs that model a large number of buffers this approach leads to problematic run-times.

In this section, we have shown that a CSDF model can lead to reduced resource requirements compared to an MRDF model and that the presented algorithm can leverage the more detailed information.

8. CONCLUSION

In this work we have presented an algorithm that determines close to minimal buffer capacities for Cyclo-Static Dataflow graphs such that the throughput requirement and constraints on maximum buffer capacities are satisfied. The buffer capacities are analytically determined from a constructed conservative schedule. Because this algorithm does not require a conversion from a Cyclo-Static Dataflow graph to a Single-Rate Dataflow graph, we do not suffer from the exponential complexity associated with this conversion and obtain an algorithm with a polynomial complexity. Related approaches do make this conversion and have excessive run-times and memory requirements for realistic Cyclo-Static Dataflow graphs.

We are currently setting up a mapping flow that determines both scheduler settings and the task to processor assignment. In order to derive a configuration that meets all constraints, we expect that this mapping flow will need to evaluate many different scheduler settings and task to processor assignments, for which the presented algorithm is an important contribution.

9. REFERENCES

[1] M. J. G. Bekooij *et al. Dataflow Analysis for Real-Time Embedded Multiprocessor System Design*, chapter 15. Dynamic and Robust Streaming Between Connected CE Devices. Kluwer Academic Publishers, 2005.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[3] G. Bilsen *et al.* Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[4] A. Dasdan. Experimental Analysis of the Fastest Optimum Cycle Ratio and Mean Algorithms. *ACM Transactions on Design Automation of Embedded Systems*, 9(4):385–418, October 2004.

[5] R. Govindarajan *et al.* Minimizing Buffer Requirement under Rate-Optimal Schedules in Regular Dataflow Networks. *Journal of VLSI Signal Processing*, 31(3), 2002.

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

[7] C. W. Mercer *et al.* Processor Capacity Reserves: Operating System Support for Multimedia Applications. In *Proc. IEEE Int'l Conference on Multimedia Computing and Systems*, pages 90–99, 1994.

[8] T. M. Parks *et al.* A Comparison of Synchronous and Cyclo-Static Dataflow. In *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, pages 204–210, October 1995.

[9] J. L. Pino and E. A. Lee. Hierarchical Static Scheduling of Dataflow Graphs onto Multiple Processors. In *Proc. IEEE Int'l Conference on Acoustics, Speech, and Signal Processing*, May 1995.

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

[11] S. Stuijk *et al.* Exploring Trade-Offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs. In *Proc. Design Automation Conference (DAC)*, pages 889–904, July 2006.

[12] M. H. Wiggers *et al.* Efficient Computation of Buffer Capacities for Multi-Rate Real-Time Systems with Back-Pressure. In *Proc. Int'l Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, October 2006.