Efficient Buffer Capacity and Scheduler Setting Computation for Soft Real-Time Stream Processing Applications

Marco Bekooij[1], Maarten Wiggers[2], and Jef van Meerbergen[3,4]
[1]NXP semiconductors, Eindhoven, The Netherlands
[2]University of Twente, Enschede, The Netherlands
[3]Philips Research, Eindhoven, The Netherlands
[4]Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract. Soft real-time applications that process data streams can often be intuitively described as dataflow process networks. In this paper we present a novel analysis technique to compute conservative estimates of the required buffer capacities in such process networks. With the same analysis technique scheduler settings can be verified. Unlike many other soft real-time analysis techniques, it is guaranteed that the desired throughput is obtained for the input stream that is used to characterize the application.

Experiments with artificial test-cases indicate that the computed FIFO capacities become more conservative if the desired throughput gets closer to the maximum throughput. The run-time of our algorithm for an H263 video decoder test-case was 14 seconds.

1 Introduction

Consumer products like smart phones and car-infotainment systems process multiple audio and video streams simultaneously. Each stream is processed by a job that can be started and stopped by the user. Jobs are typically computationally intensive and can have stringent throughput and latency requirements. For performance and power-efficiency reasons, these jobs are usually executed on an embedded heterogeneous multi-processor system.

The functional behavior of jobs that process data streams, can often be intuitively described as a YAPI process network [8]. The processes in such a network exchange high-level data structures through FIFO buffers. In [2] it has been shown that a YAPI process network can be represented as a dataflow process network [10]. The most expressive variant of a dataflow process network is the dynamic dataflow (DDF) graph [15]. The YAPI processes can be represented by the nodes in the DDF graph which are called actors.

Designers use hard real-time analysis techniques [9] for applications that must meet throughput and latency constraints and are not allowed to miss any deadline. Missing deadlines results in a steep quality reduction, like for example, clicks in the sound or visual artifacts.

The cyclo static dataflow (CSDF) model [3] can be used to show the absence of deadlock at design time, which is a requirement for hard real-time applications. The CSDF model is less expressive than the DDF model, but is considered as one of the most expressive dataflow models for which the absence of deadlocks can be shown at design time. In [3] and [18] minimum throughput analysis techniques are presented of applications that are described as a CSDF graph and are executed on a predictable multiprocessor system.

We model soft real-time applications as DDF graphs because it is not always possible to model the application as a CSDF model. This is the case if the input and output behavior of the actors are dependent on the values of the input data stream. Source decoders, such as MPEG video and audio decoders, have such an input data dependent behavior.

Soft real-time applications have a throughput and latency target. For these applications a small number of deadline misses results in a modest reduction of the quality. However, a large number of deadline misses can result in an unacceptably low quality. An MPEG video decoder is an example of a soft real-time application were the user hardly notices a sporadic frame repeat due to a deadline miss. However, a large number of successive frame repeats result in annoying hiccups in a video sequence. The use of hard real-time analysis techniques for soft real-time applications results in an over-dimensioned system because, for example, worst-case execution times are used.

The throughput and latency of a soft real-time job can be analyzed by means of simulation or with probabilistic performance analysis techniques. These techniques make a different trade-off between accuracy and run-time. The accuracy should be high enough to make useful estimates of the required scheduler settings and FIFO buffer capacities.

Simulation can be performed at different abstraction levels [5], which results in a trade-off between accuracy and simulation speed. Cycle true simulation is accurate but often impractically slow while simulation at a higher abstraction level is faster but usually has an undefined accuracy.

Probabilistic analysis techniques require that the application is represented as a marked graph [11], a Markov chain [13], or a Markov decision process [16]. The temporal behavior of the components of the system are analyzed given a representative input stream and then described with probability density functions

(PDFs). Given these PDFs, the long running average throughput is computed for the complete system. The computed throughput is an estimate with an undefined accuracy if the correlation between the execution time of the actors is ignored. The execution time of successive executions of the same actor can be correlated, and also the execution time of different actors can be correlated. This correlation can be significant in multimedia applications [11], but is often neglected [13] or approximated [11] to prevent excessive run-times of the analysis algorithm.

In the just mentioned approaches, the designer must determine the system setting of the job before he can analyze the temporal behavior. These system settings include the scheduler settings and FIFO buffer capacities. Non-linear effects and complex interactions between these settings can result in many design iterations. During each iteration, temporal analysis must be repeated. The technique presented in this paper determines the capacities of all FIFO buffers in one design iteration and thereby reduces the number of iterations.

In this paper we describe a technique that determines the FIFO buffer capacities for soft real-time applications given a desired throughput. The technique can also be used to quickly evaluate different scheduler settings. The technique is applicable for applications that are represented as the deterministic version of a DDF graph. For a given input stream it can be guaranteed that the desired minimum throughput will be obtained. The minimum throughput is specified as the minimum amount of data produced within a defined period instead of a long running average data-rate. The technique is applicable for a multiprocessor system in which resources are reserved [14].

The presented approach is based on the observation that for one input stream, the behavior of an application that is described by a DDF graph can be captured in a CSDF graph. The execution time and the number of tokens consumed and produced during one execution of a DDF actor can be represented in one phase of the corresponding CSDF actor. This results in a number of phases that is dependent on the length of the input stream, and can therefore be very large. Despite a large number of phases we can compute FIFO buffer capacities with our algorithm [18] without an excessive run-time. It is guaranteed that the computed buffer capacities are sufficient to obtain the desired throughput.

Our algorithm takes as input a CSDF graph. It computes for each input and output FIFO of an actor linear bounds on the minimum and maximum number of tokens produced and consumed per unit of time. The used bounds are similar to the service curves used in real-time calculus [12]. A significant difference with real-time calculus is that we construct a schedule for the actors such that the input and output behavior can be approximated more accurately with linear bounds. Given these linear bounds we compute a conservative estimate of the FIFO buffer capacity.

The outline of the paper is as follows. We first present in Section 2 our predictable multiprocessor system on which the soft real-time jobs are executed. Then we describe the characteristics of a CSDF graph in Section 3 and define a DDF graph in Section 4. In Section 5 we show that for a specific input stream we can analyze the temporal behavior of a DDF description of the application with a CSDF graph. The basic idea behind our algorithm is presented in Section 6. We do not present a detailed description of this algorithm because the focus of this paper is on the use of this algorithm for soft real-time applications. In Section 7 we evaluate the runtime and accuracy of our algorithm with artificial CSDF graphs. We present in Section 8 the analysis of an H263 video decoder application that is described as a DDF graph. Finally, we conclude in Section 9.

2 Predictable multiprocessor system

In this section we present the template of our multiprocessor system. In this system, resources are reserved such that the execution time of one actor is not affected by other actors. The analysis technique presented in this paper is intended for this type of systems.

Our multiprocessor template consists of tiles connected to a packet switched communication network, and is depicted in Figure 1. A tile contains a processor together with its local memory (DMEM). The instructions and data of the actors that are executed on a processor are stored in the local memory of the processor. The FIFO buffers between actors are implemented in the memories as circular buffers [6]. The communication assist (CA) in a tile copies data from a circular buffer in the local memory into a network interface FIFO or vice versa.

A processor can only access its local memory. The inter-processor communication is performed using a CA. The communication assist is configured such that it transfers data between the circular buffers in the memory and the appropriate network interface FIFO. The network is configured such that it transports the data to it destination, i.e. the appropriate FIFO in a network interface of another tile. The CA in the other tile reads the data out of the network interface FIFO and stores it into the appropriate circular buffer.

An arbiter in the CA takes care that the processor can access the memory, for example, at least 8 out of the 10 clock cycles. A fixed fraction of the remaining cycles is divided by the CA over each network interface FIFO. This arbitration policy provides a guaranteed bandwidth to the local memory for both the processor as well as the incoming and outgoing streams. Also our network [7] provides connections with a guaranteed bandwidth. These bandwidth guarantees enable the computation of the execution time of one actor in-

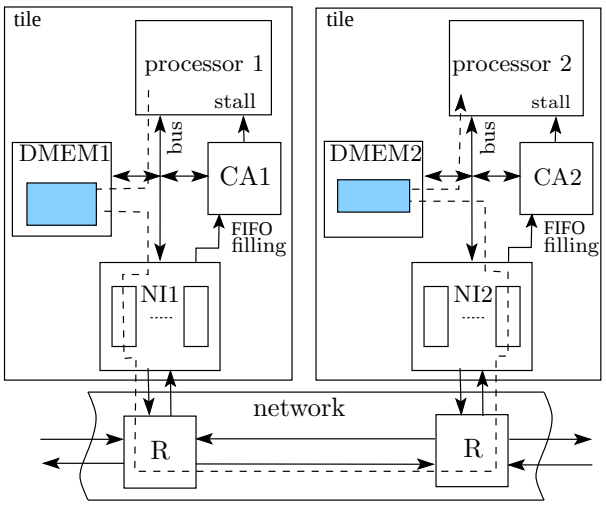

Figure 1. Template of a predictable multiprocessor system.

dependently of the behavior of the actors of other jobs. This is essential for the presented analysis technique.

3 Cyclo static dataflow model

Our algorithm uses a CSDF [3] graph as an internal dataflow model. 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 synchronize by communicating tokens over edges, and these edges 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, with $k \geq 1$, 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 $\gamma(e, ((k-1) \bmod \theta(v_i)) + 1)$ 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 $\rho(v_i, ((k-1) \bmod \theta(v_i)) + 1)$. 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 a $|E| \times |V|$ matrix, where

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

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

$$\mathbf{q} = \theta \mathbf{s} \quad \text{with} \quad \theta_{jk} = \begin{cases} \theta(v_j) & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

The repetition rate q_x of actor v_x is therefore the number of phases of v_x, within one cyclo-static period, times the relative firing frequency of the cyclo-static period. For a strongly connected and consistent CSDF graph, we can specify a desired period μ within which on average every actor v_x should fire q_x times. The throughput of the graph relates to μ^{-1}. In the remainder of this paper, we assume that the desired μ is given.

3.1 Monotonic Execution

If a CSDF graph is executed in a self-timed manner, then actors fire as soon as they are enabled. Further, a CSDF graph maintains FIFO ordering of tokens if the actors and the queues maintain FIFO ordering. Each actor maintains FIFO ordering if it has a constant response time, or has a self-cycle with one initial token. The queues maintain per definition FIFO ordering. An important property is that self-timed execution of a strongly connected CSDF graph that maintains a FIFO ordering of the tokens is monotonic in time, which is defined as follows.

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

The self-timed execution of a strongly connected CSDF graph that maintains a FIFO ordering of tokens is monotonic in time 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 and firing.

Monotonic temporal behavior of self-timed executed CSDF graphs is important because it is then sufficient to construct a schedule at design time with the desired period μ. Given monotonicity, we know that actors in the implementation will fire and produce their tokens earlier than in the at design time computed schedule.

4 Dynamic dataflow model

In this section we described a DDF graph. In Section 5 we will show that given a DDF description of the application, the capacity of the FIFO buffers can be computed with a CSDF graph.

A DDF graph is a directed graph $G_D = (V_D, E_D, \delta_D)$ that consists of a finite set of actors V_D, and a set of directed edges, $E_D = \{(v_i, v_j) | v_i, v_j \in V_D\}$. Actors synchronize by communicating tokens over edges, and these edges represent queues. The graph G_D has an initial token placement $\delta_D : E_D \rightarrow \mathbb{N}$. The firing rule of a DDF actor depends on the state of the actor, before the actor fires. The state of a DDF actor corresponds to the values of the private variables of the actor. These values can depend on data values in the tokens that are consumed during previous executions. The number of tokens consumed from a queue or produced in a queue during a firing depend also on state of the actor before the actor fires. The execution time of the i-th DDF actor execution is the difference between the i-th start-time and the i-th finish time of the actor. The execution time can depend on the state of the actor and the data in the tokens consumed during its execution.

5 DDF Analysis with a CSDF Model

In this section we show that the capacity of the FIFO buffers can be computed with a CSDF graph given a DDF description of the application. This DDF description should not contain nondeterminate merge actors [10] such that for a given input stream the computed output stream is always the same. The computed buffer capacities are large enough to obtain the desired throughput for the input stream that is used to characterize the application.

The input stream has a finite length and is processed by the actors in the DDF graph. During the processing of the input stream, each actor in the DDF graph fires a finite number of times.

The firing rule of the n-th firing of a DDF actor specifies the number of tokens that should at least be present in each input FIFO of the actor before it can fire. This firing rule depends on the values of the private variables of the actor just before it fires. These values are called the state of the actor. The state of an actor depends on the data that has been consumed during earlier firings. Therefore, the sequence of successive firing rules is completely determined by the initial state of a dataflow actor and the input streams of the DDF actor.

The number of tokens produced during the n-th DDF actor firing depends on the state of the DDF actor just before its firing and the input data consumed during the firing. The values produced during the n-th firing depend on the input data consumed during the n-th firing and the state of the DDF actor just before the n-th firing. Therefore, the streams produced by the actors are completely determinated by the input stream of the DDF graph and the initial state of the DDF actors. If the produced streams are determinate then also the sequence of successive firing rules of each DDF actor is completely determined by the input stream of the DDF graph and the initial state of the DDF actors.

The execution time of the n-th firing of a DDF actor also depends on the state of the DDF actor just before its firing and the input data consumed during the firing. A conservative estimate of the execution time of an actor firing can be obtained with an instruction set simulator on which only this particular actor is executed. We know that the input streams of a DDF actor are determinated by the input stream of the DDF graph and the initial state of the DDF actors. Therefore, the sequence of conservative execution time estimates of a DDF actor is defined by the input stream of the DDF graph and the initial state of the DDF actors.

In our multiprocessor system resources are reserved by making use of time-division multiplex (TDM) scheduling for the processors. Because we use TDM scheduling, we can compute the response time of the n-th firing of a DDF actor from the n-th execution time and the scheduler settings with equation 1, as explained in [2]. In this equation, p is the length of the TDM period, s_i the length of the time slice of actor v_i, and $\phi(v_i, f)$ the execution time of the f-th phase of actor v_i. The scheduler settings are the period p and length of the time slice s_i.

$$\rho(v_i, f) = \phi(v_i, f) + (p - s_i)\lceil \frac{\phi(v_i, f)}{s_i} \rceil \qquad (1)$$

The minimum throughput of a DDF graph for the representative input stream can be derived with a CSDF graph. This CSDF graph has the same topology as the DDF graph. The sequence of successive firing rules and execution times estimates of each DDF actor is encoded in the phases of a corresponding CSDF actor. This can result in a large number of phases of the actors.

If the FIFO capacities are given then we can determine the inverse of the minimum throughput of the CSDF graph with a maximum-cycle-mean (MCM) algorithm [4]. The CSDF graph must be transformed into an SRDF graph before the MCM algorithm can be applied. The computed MCM corresponds to the minimum interval of time in which all the phases of an actor in the CSDF graph are executed once.

The MCM algorithm assumes that the same sequence of successive firing rules and execution times of each CSDF actor is repeated forever. This is equivalent with the assumption that the same behavior of the DDF graph is continued.

However, the large number of phases in the CSDF graph usually results in a run-time of the exact MCM algorithm that is too large to be practical, as we show in Section 7. Another problem is that the buffer capacities need to be selected before the MCM can be computed.

This can lead to many iterations before suitable buffer capacities are identified that result in a desired MCM and a minimal total buffer capacity. These issues are addressed by the algorithm that is presented in the next section. This algorithm has computational complexity that is linear in the number of phases and computes sufficient and close to minimal FIFO capacity given a desired MCM. The desired MCM is equal to the period μ over which the throughput is defined.

6 Buffer Capacity Computation

In this section, we describe the basic ideas behind our algorithm that computes buffer capacities and checks scheduler settings. A detailed explanation of this algorithm can be found in [18]. The algorithm builds further upon the algorithm described in [17]. We do not present a detailed description of this algorithm because the focus of this paper is on the use of this algorithm.

Our algorithm has a computational complexity of $O(|V|^4 + |V||E|T)$ with $|E|$ the number edges, $|V|$ the number of CSDF actors and $T = \max_i(\theta(v_i))$ the maximum number of phases of the actors. The low computational complexity in the number of phases enables application of our algorithm for the computation of buffer capacities and scheduler settings given a DDF graph, a representative input stream and the desired throughput. In Section 7 we compare the run-time and accuracy of this algorithm with an exact algorithm for some examples that have a relatively small number of phases.

We will first use a simple and then a more complex example to explain the basic ideas behind our algorithm.

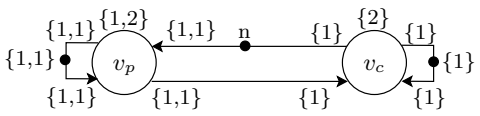

Figure 2. Producer consumer example CSDF graph.

Figure 2 depicts a CSDF graph with two actors that communicate through one FIFO buffer. This FIFO buffer is represented by the edges (v_p, v_c) and (v_c, v_p) in the CSDF graph. The number of initial tokens on the edge (v_c, v_p) is equal to the capacity of the FIFO buffer. The actor v_p has 2 phases with a response time of 1 and 2 seconds, respectively. At the beginning of their firing, actor v_p and v_c consume one token from each input edge. At the end of their firing, these actors produce one token on each output edge.

A valid schedule for the actors v_p and v_c is shown in Figure 3. The boxes in this figure denoted the intervals during which the actors are executed. The fat line s_p at right hand side of the dashed boxes denotes the as-late-as possible finish time of each firing of actor v_p given that the FIFO buffer has an infinite capacity. The line $p_l(t)$ is a linear lower bound on the number of tokens produced by actor v_p on edge (v_p, v_c). The line $p_u(t)$ is a linear lower bound of the number of tokens consumed from edge (v_c, v_p) because tokens are consumed at the beginning of a firing. Similarly, the line s_c is the as-late-as possible finish time of actor v_c under the assumption that the FIFO buffer contains at any moment in time more than one token. The lines $c_l(t)$ and $c_u(t)$ are the as-late-as possible consumption of a token by actor v_c from edge (v_p, v_c) and the as-early-as possible production of a token on edge (v_c, v_p).

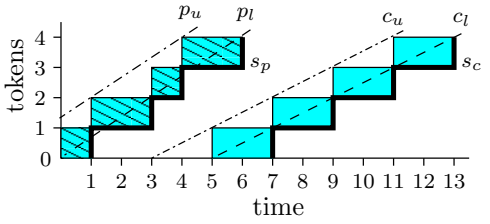

Figure 3. Producer and consumer schedule with linear bounds on token production and consumption.

It is obvious that the number of tokens consumed by actor v_c can not be larger than the number of tokens produced by actor v_p. Therefore, $\forall t, p_l(t) - c_u(t) \geq 0$ must hold.

The tokens consumed by v_p denote the amount of space acquired by v_p. The amount of tokens produced by v_c denote the amount of space released by v_c. Therefore, the FIFO buffer capacity should be larger or equal to the maximum number of tokens consumed by actor v_p minus the minimum number of tokens produced by actor v_c at any point in time, i.e. $\max_t \lceil p_u(t) - c_l(t) \rceil$. For the schedules in Figure 3 we arrive at the conclusion that an infinite FIFO buffer capacity is needed because v_p produces more tokens per unit of time than v_c consumes.

However, the schedule of actor v_p is stretched in Figure 4 such that the same number of tokens are produced as are consumed per unit of time. In this case $p_u(t)$, $p_l(t)$, $c_u(t)$, and $c_l(t)$ have the same slope. The vertical difference between p_u and c_l corresponds with a sufficient FIFO capacity. For the schedule in Figure 4, a capacity of 4 tokens is required. The required FIFO capacity becomes 2 tokens if $c_u(t)$ equals $p_l(t)$.

We will call β_{pc} the minimum distance between the start of the first firing of actor v_p and the start of the first firing of actor v_c. For the example in Figure 4, the minimum distance is 1.

The actual minimum distance between the first firings of two actors depends on the topology of the CSDF graph. Take for example the graph in Figure 5. For this graph, the minimum distances β_{01}, β_{02}, and β_{12}

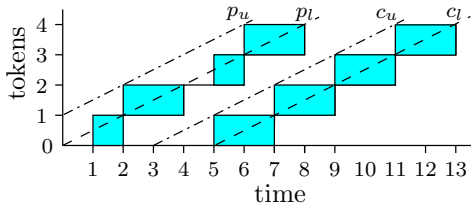

Figure 4. Stretched producer and consumer schedule.

are equal to 1. However, a minimum distance of 1 between v_0 and v_2 and a minimum distance of 1 between v_2 and v_1, results in a minimum distance of 2 between v_0 and v_1. With the dual of the minimum-cost maximum-flow network algorithm [1] we can compute in polynomial time the actual distances between the start times of each pair of actors which respects the minimum distance constraints β. Given the actual distance we can compute the capacity of the buffers. The capacity of a FIFO buffer of a producer consumer pair in the CSDF graph is made equal to $\lceil p_u(t) - c_l(t) \rceil$.

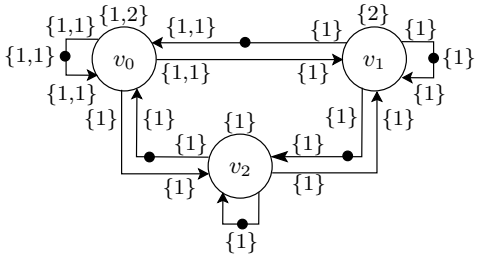

Figure 5. A CSDF graph in which the actual distance between v_0 and v_1 is a larger than β_{01}.

7 Artificial Test-Cases

In this section we evaluate the accuracy and run-time of our buffer calculation algorithm. We use three artificial test-cases that each highlight a specific aspect of the algorithm.

The first test-case is a producer consumer example, in which the response time of the producer during successive firings is uniformly distributed and the number of tokens produced and consumed is constant. For this test-case we compare the buffer capacity computed with our algorithm with the capacity computed with an exact algorithm for several desired MCM values. We compare the run-time of our algorithm with the run-time of an exact algorithm for input streams with a different length. We also evaluate whether the computed FIFO capacity is sufficient for other sequences of successive response times that have the same uniform distribution. The second test-case is also a producer consumer example, but in this test-case we vary the number of tokens consumed and produced per firing instead of the response times of the actors. For this test-case we compute the FIFO capacity for different desired MCM values. The third test-case is a chain of 3 actors. This test-case illustrates that our algorithm does not take into account that variation in the response time of an actor can be absorbed by multiple FIFOs and is not only absorbed by the in and output FIFOs of the actor.

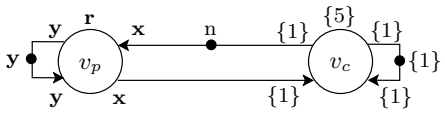

Figure 6. A CSDF graph in which actor v_p can have a variable response time or a variable production and consumption rate.

The CSDF graph of the first test-case is shown in Figure 6. The i-th element in the list **r** is equal to the response time of the i-th phase of actor v_p. The response times of the phases are uniformly distributed in the interval [1,10]. We use 1000 phases in this example. The i-th element in the list **x** and the list **y** denote the number of tokens that are consumed or produced in the i-th phase of actor v_p. All the elements in the list **x** and the list **y** are equal to one. The pair of edges (v_p, v_c) and (v_c, v_p) model a FIFO buffer. The capacity of this buffer is equal to the number of initial tokens n on the edge (v_c, v_p).

We first note that if we would determine the buffer capacity for the graph in Figure 6 with a hard real-time technique that then the desired MCM must be larger than or equal to $10 \cdot 10^3$. The MCM must be at least $10 \cdot 10^3$ because the worst-case response time of actor v_p is 10 and there are 1000 phases in one MCM period. The worst-case response time is used because another input stream could result in another sequence of successive response times. However, for an hard real-time system the desired MCM is a constraint that must be satisfied for any input stream.

By encoding the sequence of successive response times in the phases of the actors we can reduce the desired MCM to 5.5 which is equal to the absolute minimum MCM. In this example is the absolute minimum MCM equal to the average response time of actor v_p. A reduction of the desired MCM comes at the cost of a larger FIFO buffer and we cannot guarantee that a lower or equal MCM than the desired MCM is obtained for another sequence of successive response times of actor v_p. Such a sequence can occur for another input stream than the stream used to characterize the application.

The computed buffer capacity as a function of the desired MCM is plotted in Figure 7. This figure shows that the deviation between estimated capacity and the minimal capacity becomes larger if the desired MCM gets

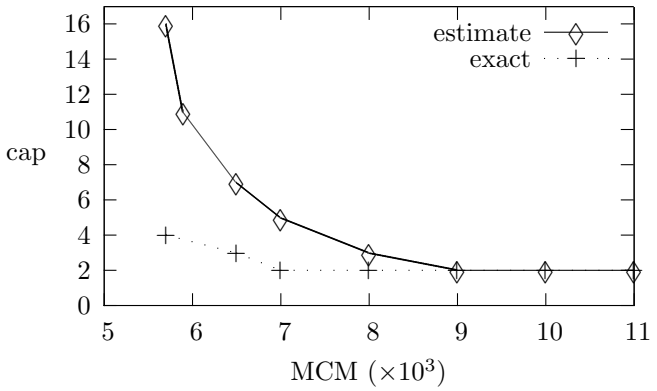

Figure 7. The estimated and minimum FIFO capacity as function of the desired MCM for the case that the response time varies.

closer to the absolute minimum MCM. The actor v_p has only 1000 phases such that we are still able to compute with a reasonable run-time the minimum FIFO capacity with an exact algorithm that is based on MCM calculation with the Howard [4] algorithm and backtracking.

The run-time of the buffer capacity computation algorithm as function of the length of the input stream for this test-case is given in Table 1. This table shows that the run-time of our algorithm grows linearly with the number of phases and the run-time of an exact algorithm grows much faster.

phases	10^2	10^3	10^4	10^5	10^6	10^7
exact	0.05s	2.09s	218s	>2h	>2h	>2h
approx	0.02s	0.09s	0.79s	7.7s	78s	780s

Table 1. Run-time of the exact and approximated buffer computation algorithm as function of the number of phases.

By changing the seed of the random generator we have generated different sequences of successive response times with the same uniform distribution. Given this sequence of response times we computed the FIFO buffer capacities with our algorithm and the exact algorithm. In both cases, we observed that the computed FIFO capacity for one stream was very rarely insufficient to obtain the same desired MCM for a different sequence of response times. This indicates that variations in the response are well absorbed by the FIFO buffers and that our algorithm takes this into account.

That variation in the response time is taken into account by our algorithm is illustrated with Figure 8. This figure shows that a peak in the response time results in a certain minimum horizontal difference between the linear bounds $p_u(t)$ and $p_l(t)$. After the peak, our algorithm takes care that the phases are started as-soon-as possible such that a successive peak will not increase the horizontal and vertical difference between the bounds. It is important that the vertical difference is not increased because this would result in a larger FIFO buffer capacity.

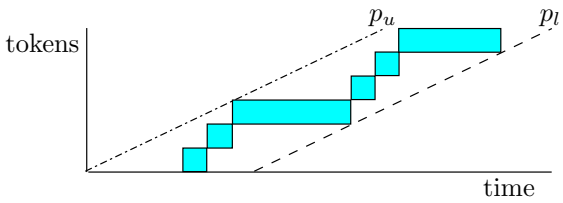

Figure 8. Absorption of peaks in the response time.

The CSDF graph of the second test-case is the same as for the first test-case and is shown in Figure 6. In this test-case all elements in the list **r** are 1. The i-th element in the list **x** denotes the number of tokens consumed and produced during the i-th firing of actor v_p. The list **r** contains uniformly distributed values in the interval [1,4]. All elements in the list **y** are equal to 1. The number of phases is 1000 such that we can compute an exact lower bound for the FIFO capacity in a reasonable time.

The computed FIFO capacity as function of the MCM is shown in Figure 9. The plot shows that the deviation between the compute minimum and estimated FIFO capacity is larger if the desired MCM gets closer to the absolute minimum MCM. This figure also shows that the exact and estimated capacities are equal to the minimum capacity of 4 for large values of the MCM. The FIFO cannot be made smaller than 4 because otherwise deadlock would occur because 4 is the maximum number of tokens produced per firing by v_p.

The third test-case is a chain of 3 actors of which the CSDF graph is shown in Figure 10. The elements in the list **r** are in the interval [1,10] and are uniformly distributed. The number of phases of v_0 is 1000 and all elements in the list **x** are 1. Given a desired MCM of 6000 we obtain with our algorithm a number of initial tokens of n equal to 9 and m equal to 2. With the exact algorithm we find n equal to 4 and m equal to 2. If we fix the maximum of n to 3 then our algorithm cannot find a solution for m anymore but the exact algorithm reports that m should at least be 3. The reason is that our algorithm does not take into account that if one FIFO does not completely absorb the variation then another FIFO can perhaps absorb the remaining variation if its capacity is large enough.

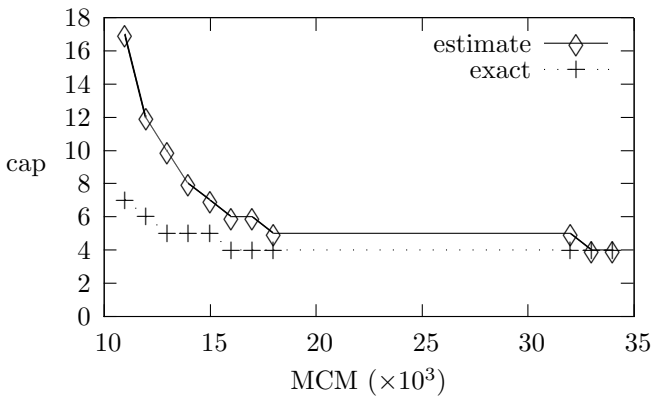

Figure 9. The estimated and minimum FIFO capacity as function of the desired MCM for varying number of tokens produced and consumed varies.

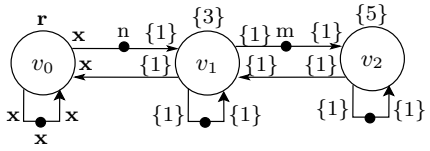

Figure 10. Chain of 3 actors in which actor v_0 has a varying response time and consumes and produces always 1 token per firing.

8 H263 Video Decoder Case-Study

In this section we describe an H263 video decoder for which we compute the capacities of the FIFOs with our algorithm. This H263 video decoder contains actors that have a response time and an input and output behavior that is input data dependent.

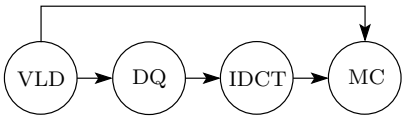

Figure 11. Process network model of an H263 video decoder.

The H263 video-decoder application is described in the C++ language as a process network (see Figure 11) using the YAPI interface. This process network description is equivalent to the DDF description in Figure 12 in which each FIFO buffer is represented by two edges because the FIFO buffers have a finite capacity.

The variable length decoding (VLD) actor in the DDF graph de-multiplexes the input stream in motion vectors, and Huffman encoded coefficients. It decodes these coefficients before it sends them as tokens with a size of 128 bytes to the de-quantization (DQ) actor. The DQ actor sends tokens with a size of 128 bytes to the inverse discrete cosine transform (IDCT) actor. The IDCT actor produces tokens with a size of 128 bytes that are send to the motion compensation (MC) actor. The MC actor receives also tokens from the VLD actor in which the motion vectors are stored. The MC actor produces tokens with a size of 38016 bytes in which one decoded frame of 176×144 pixels is stored.

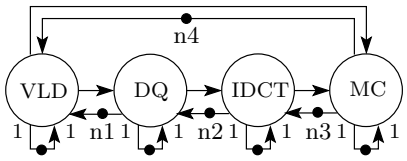

Figure 12. DDF graph of an H263 video decoder.

The H263 video-decoder application is executed on a multiprocessor instance that is according to the template in Figure 1. This instance has one ARM7TDMI processor per tile. This processor has a unified bus-interface which carries both instructions and data and does not have a cache.

The number of tokens consumed and produced per actor firing has been measured for one video sequence of 76 frames with a dataflow simulator. In order to measure the execution time we have annotated the C-code with so called "duration" statements. A value in a "duration" statement indicates the maximum number of clock cycles that are needed on the processor to execute the corresponding C-statement in the case of single cycle access latency to the local memory. We derive the duration by analyzing the assembly code and we use the debug information to locate the corresponding C-statements. In order to simplify the experiment we did not take into account that the processor is stalled if the local bus arbiter does not grant immediately access to the local memory. We also did not take into account that it cost time for the CAs to copy the data into and out of the network interface FIFOs and that it cost time to transfer data via the network. How stall cycles and communication latency can be taken into account is described in [2].

actor	phases	total exec. time (cc)
VLD	69548	$22 \cdot 10^6$
DQ	30818	$8 \cdot 10^6$
IDCT	10272	$25 \cdot 10^6$
MC	32372	$52 \cdot 10^6$

Table 2. The number of phases and total execution time of each CSDF actor.

During execution of the annotated source code on a work station, the total duration D is computed of the non-blocking code segment between two consecutive communication calls (FIFO read or write calls). The total duration D is equal to the execution time as well as the response time because in this experiment we execute one actor per processor. If actors share a processor then Equation 1 is used to compute the response time of a phase given the execution time of that phase.

desired MCM	n1	n2	n3	n4
$10 \times 52 \cdot 10^6$	3	2	8	86
$5 \times 52 \cdot 10^6$	3	2	8	85
$2 \times 52 \cdot 10^6$	5	2	12	78
$1.5 \times 52 \cdot 10^6$	6	2	43	166
$1 \times 52 \cdot 10^6$	8	2	468	984

Table 3. The computed number of initial tokens as function of the desired MCM.

The execution time and the number of tokens consumed and produced per DDF actor firing have been encoded in the phases of a CSDF graph that has the same topology as the DDF graph in Figure 12. The number of phases of each actor in this CSDF graph can be found in Table 2. There are in total 143010 phases. Given the CSDF graph, the capacity of the FIFO buffers has been computed with our buffer calculation algorithm in approximately 14 seconds on a Pentium 4, running at 2.4GHz.

The computed number of initial tokens, which are equal to the capacities of the buffers, are shown in Table 3 for different desired MCM values. As expected, the capacity increases if the desired MCM decreases. The minimum MCM is $52 \cdot 10^6$ because the MCM can not be smaller than the total execution time of one of the actors. For the minimum MCM, there is no schedule freedom for the MC actor. As a result, the token consumption schedules of the MC actor cannot be approximated well with linear bounds. However, there is schedule freedom for the VLD, DQ and IDCT actors. These schedules are adapted by our algorithm such that their token production and consumption schedules can be approximated well with linear bounds. As a result, the token production schedules of the VLD and IDCT actors do not fit well with the token consumption schedulee of the MC actor. This is shown in Figure 13 for the token production schedule of the VLD actor and the token consumption schedule of the MC actor that are used to compute $n3$. According to Table 3, this mismatch results in a significant number of initial tokens for the minimum desired MCM.

9 Conclusion

In this paper we present a technique that determines the FIFO buffer capacities for soft real-time applications

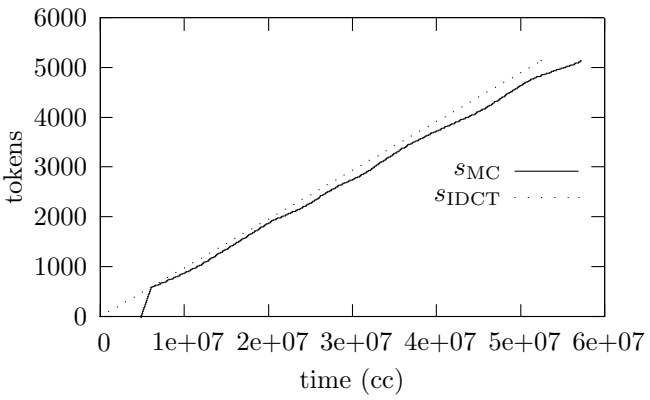

Figure 13. Token production schedules of the IDCT and MC actor.

given a desired throughput. The technique can also be used to quickly evaluate different scheduler settings, and is applicable for applications that are represented as a deterministic DDF graph. The computed buffer capacities and scheduler settings are such that for a given input stream it is guaranteed that the desired throughput will be obtained. The technique is applicable for a multiprocessor systems in which resources are reserved.

The presented technique uses a CSDF graph as an internal representation. Each CSDF actor corresponds with a DDF actor. The response time and the number of tokens consumed and produced of each non-blocking code segment is modeled as a property of a phase. Therefore, the number of phases of the CSDF actors depends on the length of the input stream and can be very large. To prevent an excessive run-time of the analysis algorithm, we conservatively approximate, for each CSDF actor, the number of tokens produced or consumed per unit of time with a linear bound. Given these linear bounds, the desired throughput, and the CSDF graph, we compute conservative estimates of the capacities of the FIFO buffers or report infeasibility.

The accuracy and run-time of our algorithm has been evaluated for artificial test-cases as well as an H263 video decoder test-case. The results of the experiments with the artificial test-cases indicate that compared with an exact algorithm the accuracy of our algorithm is lower if the desired throughput gets closer to the maximum throughput. The run-time of the buffer calculation algorithm for the H263 video decoder test-case was approximately 14 seconds. For this test-case, the total number of CSDF actor phases was 143010.

The presented technique takes the correlation between the execution time of actors into account and is suitable for any execution time distribution. It is therefore an interesting alternative for existing probabilistic analysis techniques.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] M. Bekooij, S. Parma, and J. van Meerbergen. Performance Guarantees by Simulation of Process Networks. In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2005.

[3] G. Blisen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

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

[5] A. Donlin. Transaction level Modeling: Flows and Use Models. In *Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, pages 75–80, September 2004.

[6] O. Gangwal, A. Nieuwland, and P. Lippens. A Scalable and Flexible Data Synchronization Scheme for Embedded HW-SW Shared-Memory Systems. In *Int'l Symposium on System Synthesis (ISSS)*, pages 1–6. ACM, 2001.

[7] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal Network on Chip: Concepts, Architectures, and Implementations. *IEEE Design & Test*, 22(5):414 – 421, September 2005.

[8] E. A. Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers. YAPI: Application Modeling for Signal Processing Systems. In *Proc. Design Automation Conference (DAC)*, pages 402–405, Los Angeles, June 2000.

[9] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, 1997.

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

[11] M. Li, T. van Achteren, E. Brockmeyer, and F. Catthoor. Statistical Performance Analysis and Estimation of Coarse Grain Parallel Multimedia Processing System. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 277–288, 2006.

[12] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs. In *Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, pages 128–133, September 2004.

[13] P. B. McGee, S. M. Nowick, and E. G. Coffman Jr. Efficient Performance Analysis of Asynchronous Systems Based on Periodicity. In *Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, pages 225 – 230, 2005.

[14] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Systems. In *Proc. IEEE International Conference of Multimedia Computing and Systems*. IEEE Computer Society Press, 1994.

[15] T. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, University of California at Berkeley, 1995.

[16] B. D. Theelen. *Performance Modelling for System-Level Design*. PhD thesis, Eindhoven University of Technology, 2004.

[17] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient Computation of Buffer Capacities for Multi-Rate Real-Time Systems. In *Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, 2006.

[18] M. H. Wiggers, M. Bekooij, and S. G. J. M. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. Technical Report TR-CTIT-06-70, University of Twente, November 2006.