Tight Temporal Bounds for Dataflow Applications Mapped onto Shared Resources

Hadi Alizadeh Ara*, Marc Geilen*, Twan Basten*†, Amir Behrouzian*, Martijn Hendriks† and Dip Goswami*
*Eindhoven University of Technology, Eindhoven, The Netherlands
†TNO Embedded Systems Innovation, Eindhoven, The Netherlands

***Abstract*—We present an analysis method that provides tight temporal bounds for applications modeled by Synchronous Dataflow Graphs and mapped to shared resources. We consider the resource sharing effects on the temporal behaviour of the application by embedding worst case resource availability curves in the symbolic simulation of the application graph. Symbolic simulation of the application results in a $(max, +)$ characterization matrix. This matrix specifies a set of recursive linear equations in $(max, +)$ algebra that bound the worst case execution of the application. We obtain tighter temporal bounds on the completion times of tasks than state of the art analysis. This is achieved by improving the response times of the tasks by identifying possible consecutive task executions on the resources. This enables us to use accumulated response times which are less pessimistic. Applying the new approach to real-life applications gives significant improvements over the bounds compared to state of the art.**

I. Introduction

Nowadays embedded applications are being realized on Multi-Processor Systems on Chip (MPSoCs) and share resources for cost and power reasons. These applications have real-time constraints regarding latency or throughput. One of the most important steps in designing embedded applications on shared platforms is allocating enough resources to these applications to *guarantee* their real-time constraints. Often resource allocation strategies have an iterative process in which they initially allocate resources, they analyse the temporal behaviour of the system and then they adjust resource allocation parameters based on the analysis results [1]. The temporal analysis is one of the core parts of such algorithms and since it is a part of an iterative process, it should be fast enough to make the whole allocation process practical. Sharing resources introduces uncertainties (non-determinism) to the temporal behaviour of the applications depending on the scheduling policy. For example when sharing a resource by a Time Division Multiple Access (TDMA), clock drifts cause uncertainties in the relative position of the allocated time slots which in turn causes uncertainties in the response times of the tasks. To *guarantee* that the allocated resources make an application meet its constraints, we need to obtain conservative, but tight, temporal bounds on the worst case behaviour of the system (taking into account the uncertainties) in a reasonable time. We need the bounds to be tight in order to avoid over-allocation of resources.

One of the popular methods for the temporal analysis of MPSoCs are dataflow models of computation. The dataflow model represents an application by a graph in which the nodes represent the tasks within the application and the edges model the dependencies between them. The tasks start their execution as soon as they have enough input data, then take a certain time to execute and produce output data. This is called the *self-timed* execution of the dataflow graph. The model provides a timing upper bound to the data-driven execution of the real application. Among the various types of dataflow graphs, Synchronous Dataflow Graphs (SDFG) have gained a lot of interest in modeling embedded applications. The self-timed execution of SDFG has been proven to have a periodic behaviour in terms of completion times of its tasks. One of the fastest analysis techniques to be used with SDFG is *symbolic simulation*. Symbolic analysis of applications captures the periodic behaviour of an SDFG through a set of recursive equations in $(max, +)$ algebra [2]. These equations concisely reflect the relation between the temporal bounds of the start and the end of every iteration of the application.

In our setting, the application is mapped to a multiprocessor system of which resources (processors) are being shared among several applications. We assume the resources are shared by budget schedulers to have an independent bound for each application [3]. A budget scheduler guarantees the application a minimum amount of budget (processing time) over a periodic time frame called the replenishment interval. The challenge is that in this setting the exact response times of task executions cannot be determined because the precise state of the scheduler is not known when the task starts its execution. For example, a task might be enabled at a time instance where the whole budget allocated to the application is already used for the current replenishment interval, and the task has to wait for the next replenishment interval (worst case), or the task might immediately start executing when it is enabled at the start of the allocated budget (best case). Although it is possible to obtain conservative bounds using the analysis method of [2] with worst case response times for all tasks [4], the obtained bounds are too pessimistic as shown in [5].

In this paper we adapt the symbolic analysis method of [2] to be able to analyse the applications mapped onto shared resources. We exploit the fact that the worst case response time assumption can be avoided for the busy period of the resource. We conservatively confirm busy periods of resources by symbolically identifying the sequences of immediately consecutive task executions. In contrast to existing work, in this method, response times of task executions are computed separately per each dependency of the tasks and the longest response time is used as the worst case response time. For each dependency we investigate whether it can cause a consecutive execution if it is the limiting factor for the start time of the task. This enables us to improve the response times by identifying more consecutive executions on the resources than in existing

work. We further improve the results by extending the scope of symbolic simulation and reaping the benefits of capturing consecutive executions across several iterations instead of only one.

The paper is organized as follows. Related work is discussed in Section II. Section III explains the model considered for applications. Section IV states the analysis objectives and challenges. Notational background is provided in Section V. Section VI explains the worst case execution of applications. Tighter temporal bounds for mapped applications are obtained in Section VII. Section VIII discusses temporal bounds obtained by using the characterization matrix of multiple iterations. The results obtained from real life case studies are reported and discussed in Section IX. Section X concludes the paper.

II. Related Work

Analysing the temporal behaviour of MPSoC applications is the subject of many works. The analysis methods in different works depend on the model considered for the application. We classify these models into two different groups: single-rate and multi-rate. In the single-rate models all tasks have the same input and output rate which means that they all produce and consume the same amount of data. However, multi-rate models allow different tasks to have different rates. Representative works that use single-rate models are [6]–[10]. [6] proposes an analytical method to provide tight bounds for applications modeled by a set of task graphs. The application of real-time calculus and minimum cycle mean analysis to Homogeneous Synchronous DataFlow Graphs (HSDFG) are investigated in [7] and [8] respectively. HSDFG are a subset of SDFG in which all tasks have the same rates. [10] addresses the problem of finding the critical cycle of the concurrent systems modeled by an extension of marked graphs, called signal graphs. [9] provides an algorithm to compute the exact bounds on the time separation of events for cyclic directed graphs. Application of these approaches to multi-rate models is also possible by converting these models to single-rate models. However, we provide an analysis technique that can be directly applied to multi-rate models. Applications that are multi-rate by nature, can be analysed more accurately and faster if we use analysis techniques that can be directly applied to multi-rate models.

There are several approaches that can be directly applied to multi-rate applications mapped to MPSoCs. [3], [11] embed the behaviour of the scheduler into the SDFG model of the application and use existing analysis methods for the self-timed execution of SDFG. These approaches result in an SDFG which is more complicated than the model of the application itself. Consequently, it takes much time to analyse [5]. [11] uses latency-rate servers to model the resources. This model gives pessimistic bounds unless a burst of executions are ready to be started on the resources. The method of [3] improves the bounds with respect to [11] by exploiting the fact that the completion times of consecutive iterations of tasks scheduled using budget schedules display a cyclic pattern, as long as the size of allocated slice and replenishment interval are rational numbers. The authors construct a dataflow model that precisely models this cyclic pattern, thus accurately mimicking the worst-case behavior per iteration, rather than generalizing over all iterations. However, for a range of combinations of replenishment interval and allocated budget sizes, the model is not scalable. It requires additional conservative approximations to become scalable. This might result in poor estimations on the bounds [3]. [1] provides an operational semantics for SDFG realized on deterministic shared resources. Then, it uses explicit simulation of the resources to find the periodic phase. The analysis time therefore is longer than the approaches that use a symbolic analysis method [5], [12]. The tightness of the bounds obtained by explicit simulation and symbolic simulation is similar but better than [11].

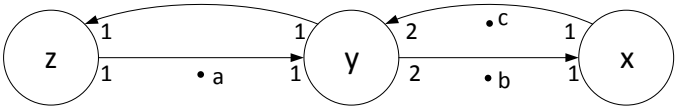

Fig. 1. An example SDFG modelling an application

The work most related to ours is [5]. The authors have combined symbolic simulation in $(max, +)$ algebra with worst case resource availability curves. Then, an approach is provided to improve the Worst Case Response Time (WCRT) of the tasks for consecutive task executions on the same resource by providing a rule that allows symbolic identification of the consecutive executions. We found that this rule is often too restrictive in the sense that it does not allow the identification of consecutive executions when the task has dependencies other than those from the previous tasks. In this paper we provide an alternative approach that more often identifies consecutive executions.

III. Application Model

A. Un-timed Model of the Application

We assume that the structure of the application i.e. the dependencies between individual tasks within the application can be abstracted by an SDFG. An SDFG is denoted by a tuple (A, C, θ, ρ). A is the set of *actors* representing the computational tasks within the application that should be carried out by the processors. $C \subseteq A \times A$ is the set of *channels*. $(a_{src}, a_{snk}) \in C$ is a first-in-first-out communication channel from actor a_{src} to actor a_{snk}. The amount of data on the channels is measured in discrete quantities called *tokens*. $\theta : C \rightarrow \mathbb{N}$ is a function that gives the initial placement of the tokens on the channels. For example, $\theta(c) = n$ means that there are n tokens on channel c. $\rho : C \rightarrow \mathbb{N} \times \mathbb{N}$ defines a tuple (r_i, r_o) for each channel where $r_i(r_o)$ is the number of tokens $a_{src}(a_{snk})$ adds (subtracts) to (from) the channel by completing (starting) a *firing*. An actor is said to be *enabled* if for all channels that end in that actor the number of tokens on the channels is at least equal to rate r_o of those channels. An enabled actor can fire and consequently produce and consume tokens on the channels. The least non-empty set of actor firings that returns the graph to its initial token placement is called an *iteration*. The *repetition vector* of the graph is a vector containing the number of firings of each actor in one iteration. The SDFG example shown in Figure 1 has three actors x, y and z and four channels $(x, y), (y, x), (z, y)$ and (y, z) with rates $(1, 2), (2, 1), (1, 1)$ and $(1, 1)$ respectively. The repetition vector is $[2; 1; 1]$ for actors x, y and z respectively.

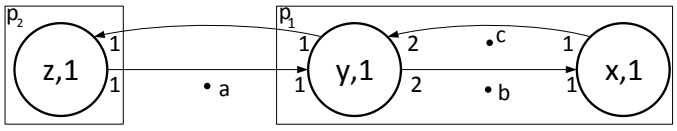

Fig. 2. An example SDFG modelling an application

B. Mapping to a Platform

An application is executed by a platform. A platform Π comprises a set P of processors. Mapping of application $g = (A, C, \theta, \rho)$ to platform Π is characterized by a tuple (β, σ). Actor binding function $\beta : A \rightarrow P$, for all actors $x \in A$ picks a processor from P that is responsible for its execution. Actor schedules σ determine the execution order of actors within the application that are bound to the same processor. Let A_p denote the set of all actors bound to processor $p \in P$ i.e. $A_p = \{x \in A \mid \beta(x) = p\}$. We give actors within A_p access to the processor by means of a *static order schedule*. A static order schedule σ for a set A of actors is a periodic infinite sequence $a_1, a_2, a_3, \cdots$ of actors in which $a_i \in A$. To have a periodic behaviour, the schedules are assumed to be free of deadlocks.

C. Timed-SDFG Model of the Mapped Application

Once the tasks within the application are bound to the processors, their execution times can be determined. Therefore we can assign *times* to the SDFG actors. A timed-SDFG of an un-timed SDFG (A, C, θ, ρ) is denoted by a tuple $(A, C, \theta, \rho, \epsilon)$ where execution time function $\epsilon : A \rightarrow \mathbb{R}^+$ assigns all tasks their execution times from the set of positive real numbers.

Assume the application shown in Figure 1 is mapped to a multiprocessor platform with two processors p_1 and p_2. Now assume that x and y are bound to p_1 and z is bound to p_2 and the static order schedules for these processors are $<x, y, x>$ and $<z>$ respectively. The execution times of all actors are assumed 1 time unit. Figure 2 depicts the timed-SDFG model of the application.

For a mapped application, the communication delay between the actors bound to different processors can also be modeled. For example a point-to-point connection with a fixed communication delay can be modeled by adding an actor with suitable execution time between communicating actors. More complex communication models such as a network-on-chip connection model of [13] can also be considered.

IV. Problem Definition

The objective of the paper is to find a mathematical model that conservatively mimics the timing behaviour of a mapped application running on a shared multiprocessor platform. Each processor can be shared between different applications using the predetermined scheduling policy of the processor. In general we assume that the platform is able to provide each application a minimum guaranteed amount of time (budget) of each processor in a given interval in which they can access the processors. For example considering a TDMA scheduler, the actors within the application can access the processors in the allocated slots. The tasks of the application are preempted when the scheduler allocates the processor to other applications and they can resume their process the next time the application has a slot. The challenge here is to obtain conservative but tight response times of the tasks and use them to model the overall timing behaviour of the system.

V. Notation

For a and b in $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$, $(max, +)$ algebra [14] considers the operations $a \oplus b \triangleq \max(a, b)$ and $a \otimes b \triangleq a + b$. For the sake of readability we use the standard max and addition notations. For $a \in \mathbb{R}_{\max}$, $\max(-\infty, a) = \max(a, -\infty) \triangleq a$ and $-\infty + a = a + -\infty \triangleq -\infty$. $\mathbb{R}^n_{\max}$ is the set of all $n \in \mathbb{N}$ dimensional vectors in $\mathbb{R}_{\max}$. Similarly, $\mathbb{R}^{n \times n}_{\max}$ is the set of all $n \times n$ matrices in $\mathbb{R}_{\max}$. Addition and max operation on vectors of the same size are applied to their corresponding elements. For $\bar{a}, \bar{b} \in \mathbb{R}^n_{\max}$, we write $\bar{a} \preceq \bar{b}$ if $\bar{a}_i \leq \bar{b}_i$ for $i = 1, \cdots, n$, where $\bar{a}_i$ denotes the i^{th} element of $\bar{a}$. Vector dot-product $\bar{a}.\bar{b}$ returns a scalar $c \in \mathbb{R}_{\max}$ where $c = \max_i(\bar{a}_i + \bar{b}_i)$. Scalar to vector addition results in a vector of the same size where the scalar is added to each of its elements.

VI. Worst Case Execution

A. Worst Case Response Times

Actors bound to the same processor, have access to the resources according to the static order schedule. Therefore the order in which the actors get access to the resources or the order of firings can be tracked. Let $\sigma_p(k)$ denote the k^{th} actor that has access to p according to the given static order schedule σ. Let tuple (p, k) denote firing k on processor p which corresponds to actor $x = \sigma_p(k)$. From the moment firing (p, k) becomes enabled i.e. actor x becomes able to start firing, it requires $\epsilon(x)$ units of processing time to be completed where $\epsilon(x)$ is the execution time of actor x. Since preemption is allowed by the platform, the actor completes its firing when the sum of the processing time that it gets after enabling time, exceeds $\epsilon(x)$. The total time it takes for an actor to complete its firing once it is enabled, is called the *response time* of the firing. An upper bound on the completion time of the actor firing can be calculated from the time instance it becomes enabled as

$$c(p, k) = e(p, k) + \omega(x) \tag{1}$$

where $c(p, k)$ denotes an upper bound on the completion time of (p, k), $e(p, k)$ denotes its enabling time and $\omega(x)$ denotes the Worst Case Response Time (WCRT) of actor x. To calculate $\omega(x)$, we have to find the minimum amount of time that guarantees the application $\epsilon(x)$ units of processing time. For example, let's consider processor p shared by example TDMA in Figure 3. Now consider the first firing on processor p, i.e. $(p, 1)$ which is of actor x for example, with $\epsilon(x) = 1$. Assume it becomes enabled at $e(p, 1)$. The worst case enabling time with respect to the positioning of the allocated slots is depicted in the same figure. In this situation, the actor has to wait 4 time units to start processing at the next allocated slot; hence, $\omega(x) = 5$ and in the worst case it will completely be processed at $c(p, 1) = e(p, 1) + 5$. This means that 5 time units is the minimum time that guarantees the application 1 unit of processing time using the example TDMA.

To find the WCRTs of actor firings with different execution times, we use the notion of Worst Case Resource

Curve (WCRC). The WCRC is the same as the lower bound service curves of real-time calculus [15]. This curve specifies the minimum amount of service (processing time) that the application is guaranteed to get in any time interval [5].

Definition 1 (Worst Case Resource Curve): A WCRC $\zeta(\delta)$ specifies the minimum amount of processing time allocated to the application in every time interval of length δ.

For a TDMA scheduler for example, the WCRC has a periodic behaviour of length w where w is the size of the time wheel i.e. $\zeta(\delta + w) = \zeta(\delta) + \Delta$ for some constant Δ. The first period of the WCRC can be calculated by

$$\zeta(\delta) = \min_{t \in [0,w)} \int_{t}^{t+\delta} \alpha(u)du$$

where $\alpha(t)$ is 1 if at time t the processor is allocated to the application and 0 otherwise. The WCRC of the example TDMA scheduler is depicted in Figure 3. According to this figure, the worst case response time of a firing is the earliest time that the processor dedicates the required processing time to the actor. Therefore the WCRT of a firing actor can be obtained from the WCRC of the processor by the following equation

$$\omega(a) = \inf\{\delta \in \mathbb{R}^+ \mid \zeta(\delta) \geq \epsilon(a)\} \tag{2}$$

where $\inf(.)$ is the infimum operation. As an example, a firing with execution time of 1 has WCRT of 5 and execution time 2 has WCRT of 7 time units. As explained, using WCRC to find the completion time of the firings assumes the longest waiting time for the actor; i.e. it starts from the beginning of the curve where the processor is not available for the application. It is well known that this assumption can be avoided for time intervals in which we can ensure the processor will be kept busy within those time intervals. Such time intervals are called the *busy periods* of the processor [16]. Within a busy period, the first firing starts from the beginning of the curve and the next firings that follow, continue on the same curve because they can start as soon as the processor completes the previous firing. One way to be sure that the processor is kept busy is when a sequence of *consecutive* firings are enabled. A sequence of firings is said to be consecutive if we know that for all its firings, their enabling is no later than the completion of the previous firing.

Definition 2 (Consecutive Firings): A sequence of firings $R = (p,k), \cdots, (p,k+n)$ is said to be consecutive if the enabling time of firing (p,i) is guaranteed to be at most the completion time of firing $(p,i-1)$ for $k+1 \leq i \leq k+n$.

For example, let's assume p has completed the first firing and from the properties of the application holds that the second and third firings on p enable once the previous firing completes i.e. $e(p,i) = c(p,i-1)$ for $i = 2,3$. Let's also assume the actors y and x with execution times of 1 correspond to these firings respectively. The second firing can start at the same time as the first firing completes. Therefore it starts from the point on the curve where $(p,1)$ completed, and hence it holds that $c(p,2) = e(p,1) + 7$. Similarly, the third firing also continues the same curve and $c(p,3) = e(p,1) + 8$ as shown in Figure 3. Note that if we fail to exploit the fact that these firings are consecutive, we obtain the pessimistic upper bound of 15 time

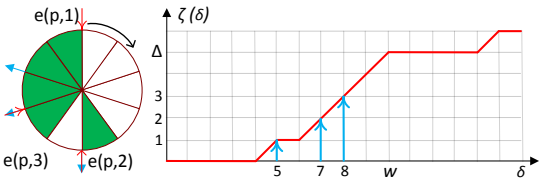

Fig. 3. An example TDMA and its corresponding WCRC. Only the green-coloured slots are allocated to the application.

units, since Equation 2 returns the WCRT of 5 time units for all actors and the firings start one after another.

From the example one can realize that the completion time of firings within a sequence of consecutive firings are computed from the enabling time of the first firing and the WCRT for a task with the same execution time as the sum of the execution times of the individual actors. Now we adapt Equations 1 and 2 to obtain tight upper bounds for the completion times of firings in busy periods. Let R_i denote a subsequence of R with the first i firings and $[R]_i$ denote the i^{th} firing in R. An upper bound on the completion time of firing $[R]_i$ can be calculated by adding *the accumulated worst case response time*, $\omega(R_i)$ to the enabling time of the first firing in R i.e.

$$c([R]_i) = e([R]_1) + \omega(R_i). \tag{3}$$

$\omega(R_i)$ is equal to the WCRT of an execution time equal to the sum of execution times of the corresponding actors of the firings in R_i i.e.

$$\omega(R_i) = \inf\{\delta \in \mathbb{R}^+ \mid \zeta(\delta) \geq \sum_{l=1}^{i} \epsilon([R_i]_l)\}. \tag{4}$$

In the following, we will identify the busy periods and obtain tight upper bound on the completion times of the firings based on the above equations.

B. Worst Case Actor Firing Semantics

An actor in the next position of the static order schedule can start firing when the processor and the tokens consumed by the actor are available. Therefore, the start time of the firing is synchronized with the latest time that the processor and the consumed tokens are available.

To illustrate, consider the timed-SDFG shown in Figure 2. Lets assume p_1 is shared by the example TDMA and p_2 is shared by another TDMA of the same wheel size in which there is always an allocated slot next to an unallocated slot as shown in Figure 4. Let t_a, t_b, t_c denote the availability times of the initial tokens labelled a, b, c in Figure 2 and t_{p_1}, t_{p_2} denote the initial availability times of processors p_1, p_2 respectively. We compute $c(p,k)$ for firings involved in one iteration of the graph given that $t_a = t_b = t_c = t_{p_1} = t_{p_2} = 0$.

z is the first actor in the static schedule of p_2 but it is not enabled initially because it does not have enough tokens on channel (y,z). The first firing on p_1, i.e. $(p_1,1)$, is of x, which is dependent on the availability of token b and on p_1. Thus, it completes at the latest at $c(p_1,1) = \max\{t_b, t_{p_1}\} +$

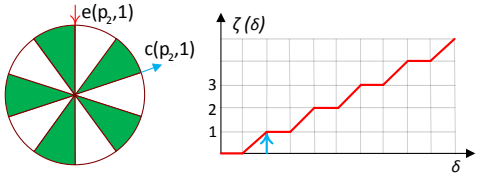

Fig. 4. The TDMA scheduler used on Processor p_2 and its corresponding WCRC

$\omega(\sigma_{p_1}(1)) = 0 + \omega(x) = 5$. By completion of this firing, a token is produced on channel (x, y) and the availability time of p_1 for the next firing is $c(p_1, 1)$. The availability of p_1 and the token produced on (x, y) together with tokens a and c are required for the second firing on p_1 which is of y. Therefore, $c(p_1, 2) = \max\{t_c, t_a, c(p_1, 1)\} + \omega(\sigma_{p_1}(2)) = \max\{0, 0, 5\} + \omega(y) = 5 + 5 = 10$. Now, z is enabled and x is enabled for the second time. $(p_2, 1)$ is a firing of z and requires the availability of p_2 and the token produced on channel (y, z) by y. Thus, $c(p_2, 1) = \max\{t_{p_2}, c(p_1, 2)\} + \omega(\sigma_{p_2}(1)) = \max\{0, 10\} + \omega(x) = 10 + 2 = 12$. By the completion of $(p, 3)$ at $c(p_1, 3) = \max\{c(p_1, 2), c(p_1, 2)\} + \omega(\sigma_{p_1}(3)) = 10 + 5 = 15$ the iteration is complete.

C. Symbolic Simulation & Temporal Bounds

Next, we simulate the graph *symbolically*. That is, we consider symbols for the availability times of initial tokens and initial availability times of processors as opposed to the concrete simulation we did above. This way we can represent the production time of a token for example, in a form called *symbolic time stamp*, that contains all time differences between the production time of the token and the availability time of initial tokens. This enables us to find relations between the start and completion times of actors within one iteration that hold for any arbitrary iteration.

Following [12], the production times of each token and consequently the start and the completion times of each actor firing can be represented by a symbolic time stamp of the form $t = \max_i(t_i + g_i)$ where t_i are the availability times of initial tokens (a, b, c) and initial availability times of processors (p_1, p_2). $g_i \in \mathbb{R}_{\max}$ are suitable constants that indicate the minimum time difference between t and t_i if t_i is a limiting factor in production (completion/start) of the token (actor firing). It is $-\infty$ if there is no dependency between t and t_i. In the $(max, +)$ algebra domain we can represent such a symbolic time stamp with the vector dot-product i.e. $t = \bar{t}.\bar{g}$ where $\bar{t} = [t_a; t_b; t_c; t_{p1}; t_{p2}]^T$ and $g \in \mathbb{R}^5_{\max}$ for the example. Using the time stamps, the completion time of $(p_1, 2)$ and consequently the production times of the tokens produced by it, $c(p_1, 2) = \max(t_a + 5, t_b + 10, t_c + 5, t_{p_1} + 10)$ can be represented as $[5; 10; 5; 10; -\infty]^T$. This vector representation of the start (completion) time of firings is called the *symbolic start (completion)* time and it is denoted by $\bar{e}(\bar{c})$. Another important interpretation of the elements of the symbolic time stamp $t = \max_i(t_i + g_i)$, is that if of all limiting factors of t, t_j is the actual limiting factor i.e. $j = \text{argmax}(t_j + g_j)$, g_j is the *maximum* time difference between t and t_j. For example, the maximum time difference between $c(p_1, 2)$ and t_b is equal to the second element of $\bar{c}(p_1, 2)$ i.e. 10, if we know that the availability of b is limiting the completion of $(p_1, 2)$. For symbolic simulation of the example, let $\bar{t}_a, \bar{t}_b$ and $\bar{t}_c$ denote the following symbolic time stamps: $[0; -\infty; -\infty; -\infty; -\infty]^T$, $[-\infty; 0; -\infty; -\infty; -\infty]^T$ and $[-\infty; -\infty; 0; -\infty; -\infty]^T$ respectively. Similarly, let $\bar{t}_{p_1}$ and $\bar{t}_{p_2}$ denote the initial availability time of the processors and correspond to the time stamps $[-\infty; -\infty; -\infty; 0; -\infty]^T$ and $[-\infty; -\infty; -\infty; -\infty; 0]^T$ respectively. Now we can simulate the example symbolically. The first firing on p_1 starts at $\bar{e}(p_1, 1) = \max\{\bar{t}_b, \bar{t}_{p_1}\} = [-\infty; 0; -\infty; 0; -\infty]^T$ and completes the latest at $\bar{c}(p_1, 1) = [-\infty; 0; -\infty; 0; -\infty]^T + 5 = [-\infty; 5; -\infty; 5; -\infty]^T$. Similarly the symbolic completion times of other firings are determined as $\bar{c}(p_1, 2) = [5; 10; 5; 10; -\infty]^T, \bar{c}(p_1, 3) = [10; 15; 10; 15; -\infty]^T$ and $\bar{c}(p_2, 1) = [7; 12; 7; 12; 2]^T$. At the end of the iteration, the symbolic production times of the tokens remaining on the graph will be regarded as the availability times of the initial tokens from the point of the next iteration. Moreover, the completion times of the last firings on the processors will be regarded as the initial availability times of the processors. For the next, let $\bar{t}' = [t'_a; t'_b; t'_c; t'_{p_1}; t'_{p_2}]^T$ denote the new availability times of the tokens and processors. From [5], the relation between $\bar{t}'$ and $\bar{t}$ can be found by collecting the symbolic time stamps of the tokens remaining on the graph at the end of the iteration and the time stamps of the last firings on the processors. This relation is determined by the $(max, +)$ characterization matrix M.

$$\begin{pmatrix} t'_a \\ t'_b \\ t'_c \\ t'_{p_1} \\ t'_{p_2} \end{pmatrix} = \begin{pmatrix} 7 & 12 & 7 & 12 & 2 \\ 5 & 10 & 5 & 10 & -\infty \\ 10 & 15 & 10 & 15 & -\infty \\ 10 & 15 & 10 & 15 & -\infty \\ 7 & 12 & 7 & 12 & 2 \end{pmatrix} \cdot \begin{pmatrix} t_a \\ t_b \\ t_c \\ t_{p_1} \\ t_{p_2} \end{pmatrix}$$

M is obtained by noticing that a, b and c are produced when $(p_2, 1), (p_1, 2)$ and $(p_1, 3)$ completed respectively. Thus, $t'_a = \bar{c}(p_2, 1)$, $t'_b = \bar{c}(p_1, 2)$ and $t'_c = \bar{c}(p_1, 3)$. The last firings on processors are $(p_1, 3)$ and $(p_2, 1)$. Thus $t'_{p_1} = \bar{c}(p_1, 3)$ and $t'_{p_2} = \bar{c}(p_2, 1)$. Using M we can compute an upper bound for any arbitrary iteration with the recurrent relation $\bar{\gamma}_{k+1} = M \cdot \bar{\gamma}_k$ where $\bar{\gamma}_k \in \mathbb{R}^n_{\max}$ is a vector of time stamps that marks an upper bound to iteration k. With this recurrence, the *cycle time* of the application i.e. the average rate in which the iterations of the application complete can also be computed. An upper bound on the cycle time can be obtained by calculating the $(max, +)$ eigenvalue of M [2]. For this example we obtain the cycle time of 15.

VII. Tighter Temporal Bounds

In [5] an approach has been developed to provide tighter upper bounds on the completion times of firings. This approach sets up a condition to identify consecutive firings on the processors symbolically in order to avoid using WCRTs for each individual firing. To calculate the symbolic completion time of a firing, the authors initially assume that it will start a new sequence of consecutive firings on the processor, then if the symbolic start time of the next firing is not greater than the symbolic completion time of the firing, i.e. $\bar{e}(p, k + 1) \preceq \bar{c}(p, k)$, then the next firing is appended to the sequence of consecutive firings. Then the accumulated worst case response time can be used to calculate its completion

time based on Equation 3. Otherwise it is assumed to start a new sequence of firings and therefore, its completion time is calculated by Equation 1. For our example, execution of actor x starts a sequence of consecutive firings on p_1 at $\bar{e}(p_1,1) = [-\infty;0;-\infty;0;-\infty]^T$. It completes at $\bar{c}(p_1,1) = \bar{e}(p_1,1) + \omega(x) = [-\infty;0;-\infty;0;-\infty]^T + 5 = [-\infty;5;-\infty;5;-\infty]^T$. The next firing can start at the symbolic time stamp $\max\{\bar{t}_a, \bar{t}_c, \bar{c}(p_1,1)\} = [0;5;0;5;-\infty]^T$. Since $\bar{e}(p_1,2) = [0;5;0;5;-\infty]^T \not\preceq \bar{c}(p_1,1) = [-\infty;5;-\infty;5;-\infty]^T$, the second firing on p_1 also starts a new sequence of consecutive firings and completes at $\bar{e}(p_1,2) + \omega(y) = [0;5;0;5;-\infty]^T + 5 = [5;10;5;10;-\infty]^T$. The third firing on p_1 starts at $[5;10;5;10;-\infty]^T$. Since $\bar{e}(p_1,3) \preceq \bar{c}(p_1,2)$ the completion of the third firing is calculated using Equation 3. Therefore, for this firing, $\bar{c}(p_1,3) = \bar{e}(p_1,2) + \omega(y \cdot x) = [7;12;7;12;-\infty]^T$. We have obtained a tighter upper bound on the completion time of $(p_1,3)$. This also affects the upper bound on the cycle time of the application, which reduces from 15 to 12. This method of symbolically identifying consecutive firings has been proposed in [5]. Next we present our symbolic identification of consecutive firings.

We show that condition $\bar{e}(p,k+1) \preceq \bar{c}(p,k)$ is too restrictive. It requires *all* elements of $\bar{e}(p,k+1)$ not to be greater than $\bar{c}(p,k)$ and if this condition is not satisfied then the WCRT is used according to Equation 1. This creates overly pessimistic dependencies between the availability times of the tokens/processors and the completion times of the firings. For example recall that $\bar{e}(p_1,2) = \max\{\bar{t}_a, \bar{t}_c, \bar{c}(p_1,1)\} \not\preceq \bar{c}(p_1,1)$ since

$$\max\left\{\begin{pmatrix}0\\-\infty\\-\infty\\-\infty\\-\infty\end{pmatrix},\begin{pmatrix}-\infty\\-\infty\\0\\-\infty\\-\infty\end{pmatrix},\begin{pmatrix}-\infty\\5\\-\infty\\5\\-\infty\end{pmatrix}\right\} = \begin{pmatrix}0\\5\\0\\5\\-\infty\end{pmatrix} \not\preceq \begin{pmatrix}-\infty\\5\\-\infty\\5\\-\infty\end{pmatrix}.$$

This means that every time an initial token is consumed by actors, the condition is not met. Therefore for such cases we always have to use Equation 1. In this example it leads to

$$\bar{c}(p_1,2) = \begin{pmatrix}0\\5\\0\\5\\-\infty\end{pmatrix} + 5 = \begin{pmatrix}5\\10\\5\\10\\-\infty\end{pmatrix}$$

The second element of $\bar{c}(p_1,2)$ is 10. This expresses that there is at most a time difference of 10 time units between t_b and $c(p_1,2)$ when b is limiting the completion. This however is too pessimistic because in case b is indeed the limiting factor, then $(p_1,2)$ will immediately start when $(p_1,1)$ completes; we can conclude that in this situation, $(p_1,1)$ and $(p_1,2)$ are consecutive. Therefore the maximum time difference between t_b and $(p_1,2)$ cannot be more than the accumulated response time of $\omega(z \cdot y) = 7$. However, if t_c or t_a is the limiting factor then the maximum time difference between t_a and $(p_1,2)$ or t_c and $(p_1,2)$ can be at most the WCRT of $\omega(y) = 5$. This means that the symbolic completion time of $(p_1,2)$ should be $[5;7;5;7;-\infty]^T$. To avoid such pessimistic completion times, we propose to compute the completion time of each firing with respect to each t_i, separately.

To be able to find the maximum time differences between the completion times of firings and all t_i separately, we have to find all dependencies between firings for the firings involved in one iteration of the application graph. This enables us to separately capture all possible dependency paths that connect the completion times of firings to all t_i. Then for each dependency path we can separately determine which firings are consecutive in it. To do so, we first find all firings on which the start of a firing directly depends.

Definition 3 (Dependency Set): The dependency set $D_{(p,k)}$ comprises the firings the completions of which are required for (p,k) to start, i.e. $D_{(p,k)} = \{(p',k') \mid (p,k)$ consumes a token produced by $(p',k')\} \cup \{(p,k-1)\}$.

Note that the first firing on the processor i.e. $(p,1)$ is dependent on the initial availability of p which can be denoted by $(p,0)$ as if it was released by completion of firing 0 on p. In addition, to be able to represent the dependency on the availability of initial tokens, we represent them as if they were produced by completion of firings with negative indices on an arbitrary processor i.e. (p,k) with $k<0$. The firings with non positive indices are referred to as *initial dependencies*.

During symbolic simulation, for each token that is produced by firings, in addition to the symbolic time stamp that shows its production time, we add extra information regarding the firing that produced them. By keeping track of the tokens produced and consumed by firings, we can extract the dependency sets of firings. Figure 5 shows the dependency graph associated with the execution of the example SDFG for two subsequent iterations. It is obtained by simulating the graph and finding the dependency set of each firing during the simulation. In this graph, the nodes represent the firings. A directed edge from (p',k') to (p,k) indicates that (p,k) is dependent on (p',k'). The black edges indicate the firing dependencies on the same processor and the red edges indicate the dependencies on different processors. For the sake of readability, the initial dependencies are given their own names instead of firings with non positive indices. Using this graph we can track the dependencies of each firing back to the initial dependencies and separately compute the time difference between them. The key point is that if two nodes are connected only by black edges, then the time difference between them is equal to the accumulated worst case response times of all firings between them including the last node. For example the time difference between $(p_1,3)$ and t_b (b in the figure) is equal to $\omega(z \cdot y \cdot z) = 8$ which is less pessimistic compared to 12 obtained by the method of [5]. However, if the connecting path contains red edges, we have to separate the path on the red edges. For each separate piece of the path then, we can use the accumulated response times to compute the partial time differences. Finally, the response time is equal to the sum of all partial time differences. For example the time difference between $(p_2,1)$ and t_b is equal to $\omega(z \cdot y) + \omega(x) = 7+2 = 9$. If there is more than one path between two nodes, then the time difference is equal to the maximum of time differences in all paths. For example there are two different paths between $(p_2,2)$ and t_{p_2} (p_2 in the figure). The maximum time difference between them is $\max\{\omega(x \cdot x), \omega(x) + \omega(y) + \omega(x)\}$– the first term for the path with black edges and the second term for the path with the red edges.

Algorithm 1 sketches the construction of the character-

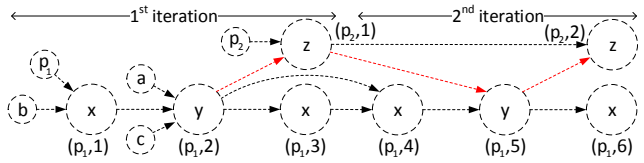

Fig. 5. Dependency graph associated with the execution of the example SDFG for two iterations

ization matrix based on separate analysis per dependency for a given timed-SDFG application graph g, platform Π, binding function β, WCRCs and static order schedules ζ and σ per processor. The algorithm stores the dependency sets and completion times of all firings in sets D and Φ respectively. D and Φ are computed following a symbolic execution of an iteration of the graph. D is initially empty and Φ is initialized with the completion times of initial dependencies (lines 2,3). The repetition vector is calculated in line 4. Then, the algorithm picks an enabled actor (line 7). The processor which the enabled actor is bound to is found from β in line 8. Then the firing counter on this processor is updated (line 9). Initially, it is assumed that the firing is not part of any previously started consecutive firing sequences; thus $R = (p, k)$ (line 10). The dependency set of the firing is constructed in line 11 and added to D in line 12. Line 13 calls Algorithm 2 to compute the upper bound on the symbolic completion time of the firing. This algorithm first constructs the dependency graph. Starting from node (p, k), it connects all nodes representing the firings in the dependency set of (p, k) to this node. Then, the same action is taken for each node in the dependency set of (p, k) only if it represents a firing on the same processor. This process continues until all source nodes of the graph (the ones without input edges) are either representing initial dependencies or firings on other processors. Then, the symbolic completion time of the firing is obtained by adding the maximum time difference of all paths that connect the source nodes to (p, k) i.e. the accumulated response time of all firings in the path, to the symbolic completion time of firings represented by source nodes and taking the maximum of all. The calculated symbolic completion time of the firing is added to the set of completion times (line 14). When all firings involved in one iteration of the graph are stored in Φ, the algorithm terminates after collecting the completion times of tokens left on the graph and of the last firings on the processors (line 17).

Algorithm 2 computes the upper bound on the symbolic completion times of firings in a recursive fashion from its dependencies. The inputs of this algorithm are: firing (p, k), set Φ of all symbolic completion times of stored firings, set D of their stored dependency sets, sequence R of the consecutive firings and the WCRCs of the processors. Initially $\bar{c}(p, k)$ is set to a vector of appropriate size with all of its elements equal to $-\infty$ (line 2). The algorithm picks a dependency from $D_{(p,k)}$ (line 4). A symbolic completion time for (p, k) is computed form this dependency as follows: if the dependency is either initial i.e. (p', k') s.t. $k' \leq 0$ or not from p i.e. (p', k') s.t $p' \neq p$, the symbolic completion time from this dependency is calculated by adding $\omega(R)$ to the symbolic completion time of that dependency (line 6); if the dependency is such that

Algorithm 1 Construct the $(max, +)$ matrix of a SDFG

1: $ConstructMaxPlusMatrix(g, \Pi, \beta, \zeta, \sigma)$
2: $D \leftarrow \emptyset$
3: $\Phi \leftarrow$ initial dependencies
4: $\bar{r} \leftarrow$ compute the repetition vector of g
5: $\bar{r}' \leftarrow \bar{r}$
6: **while** $\bar{r}' \neq$ zero vector **do**
7: $a \leftarrow$ pick an enabled actor such that $\bar{r}'(a) \geq 0$
8: $p \leftarrow \beta(a)$
9: $k \leftarrow$ update the firing counter on $p = \beta(a)$
10: $R \leftarrow (p, k)$
11: $D_{(p,k)} \leftarrow$ construct the dependency set for (p, k)
12: $D \leftarrow D \cup \{D_{(p,k)}\}$
13: $\bar{c}(p, k) =$ WCCTF$((p, k), D, \Phi, R, \zeta)$
14: $\Phi \leftarrow \Phi \cup \{\bar{c}(p, k)\}$
15: $\bar{r}' \leftarrow \bar{r}' - 1$
16: **end while**
17: $M \leftarrow$ collect the new time stamps of initial tokens and processor availabilities from Φ

Algorithm 2 Worst Case Completion times of firings

1: **procedure** WCCTF$((p, k), D, \Phi, R, \zeta)$
2: $\bar{c}(p, k) \leftarrow [-\infty; \cdots; -\infty]$
3: **while** $D_{(p,k)} \neq \emptyset$ **do**
4: $(p', k') \leftarrow$ pick a dependency form $D_{(p,k)}$
5: **if** $p' \neq p \vee k' \leq 0$ **then**
6: $\tilde{\bar{c}}(p, k) \leftarrow \bar{c}(p', k') + \omega(R)$
7: **else**
8: $R \leftarrow$ prepend (p', k') to R
9: $\tilde{\bar{c}}(p, k) \leftarrow$ WCCTF$((p', k'), D, \phi, R, \zeta)$
10: **end if**
11: $\bar{c}(p, k) = \max(\tilde{\bar{c}}(p, k), \bar{c}(p, k))$
12: $D_{(p,k)} \leftarrow D_{(p,k)} \setminus (p', k')$
13: **end while**
14: **end procedure**

$p' = p$, (p', k') is prepended to R i.e. $R = (p', k'), (p, k)$ (line 8) and Algorithm 2 is invoked recursively for (p', k') and new R to obtain the symbolic completion time from this dependency (line 9). The symbolic completion time of (p, k) is updated to the maximum symbolic completion time from the dependency and itself (line 11). The algorithm terminates when the completion times from all dependencies in all invocations are computed.

VIII. Matrix of Multiple Iterations

In obtaining the characterization matrix by symbolic simulation of the graph for one iteration, we treated the completion times of tokens and the last firings on processors at the end of the iteration, as the initial dependencies for the next iteration. This implies that no consecutive firings can be identified across multiple iterations. However, by continuing the simulation for more than one iteration, we can keep track of consecutive firings for firings involved in the next iterations. This indeed improves the upper bounds on the completion times of firings for the next iterations. Let's assume we want to compute the time difference between the completion time of $(p_1, 4)$ in the second iteration and the availability of b. Figure 5 shows a time difference equal to $\omega(z \cdot y \cdot z \cdot z) = 9$. However, by stopping the simulation at the end of the first iteration, the time

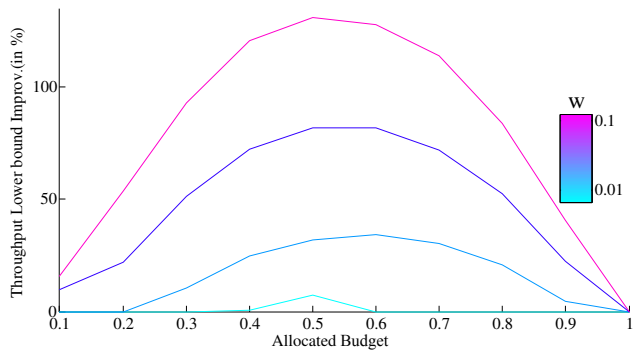

Fig. 6. Average improvements in throughput lower bounds compared to [5]

difference can only be calculated by adding $\omega(z) = 5$ to the time difference between b' and b i.e. $\omega(z \cdot y \cdot z) = 8$ which gives the time difference of 13 time units. In our example the cycle time obtained by constructing the matrix for two iterations is $7\frac{1}{2}$ and for three it reduces further to $6\frac{1}{3}$. Note that extending the scope of simulation comes at the cost of increased analysis time.

IX. Evaluation

We have implemented our temporal analysis method in the SDF3 tool [17]. We compared the throughput (1/cycle time) lower bound obtained by our approach with the state of the art analysis of [5] for three real life applications: H.263 encoder, H.263 decoder and sample rate converter, all available in the SDF3 tool. To have a fair comparison in terms of analysis run-time, we do not analyse across iterations in this comparison. For each application, we used SDF3 to map it to a multiprocessor platform with four processors such that the total work load is evenly distributed between processors as much as possible. We limit the replenishment intervals to $0.01 \times C_f \leq w \leq 0.1 \times C_f$ where C_f is the cycle time of the application when all processors are fully allocated to the application. Large replenishment intervals cause huge delays in execution of the application, which is not desired; small replenishment intervals are less useful because of the context switch overhead. Figure 6 shows the average relative improvements in the throughput lower bounds of applications for different replenishment intervals and allocated budgets. For each combination of replenishment interval and allocated budget, the $(max, +)$ characterization matrix is constructed by Algorithm 1. Then a lower bound on throughput is computed by obtaining the $(max, +)$ eigenvalue of this matrix. As shown in the figure, the improvement ratio decreases when the application gets smaller or larger processor shares. In these cases using the accumulated worst case response times does not have much improvements over WCRTs. The average analysis run-time for the mentioned applications on a standard computer is 320 milliseconds which is 17% longer compared to [5]. It is still in the practical range.

X. Conclusion

We have proposed an analysis technique that provides conservative, but tight temporal bounds for SDFG applications running on shared resources. Our approach uses WCRCs to obtain the WCRT of the tasks and then uses them in the symbolic simulation of the application graph. It results in the $(max, +)$ characterization matrix that mimics the worst case execution of the application. We obtain tighter bounds than the state of the art by identifying consecutive firings on the processors. For consecutive firings, we are able to use the accumulated worst case response times that are less pessimistic. By identifying consecutive firings across iterations, we obtain even tighter bounds. This analysis can be extended to give tighter bounds for dynamic dataflow models such as Scenario Aware Dataflow.

Acknowledgment

This research is supported by the ARTEMIS joint undertaking through the ALMARVI project (621439).

References

[1] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Proc. 44th annual Design Automation Conference*, 2007.

[2] M. Geilen, "Synchronous dataflow scenarios," *ACM Transactions on Embedded Computing Systems*, 2010.

[3] A. Lele, O. Moreira, and P. J. Cuijpers, "A new data flow analysis model for TDM," in *Proc. 10th ACM international conference on embedded software*, 2012.

[4] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J. D. Mol, S. Stuijk, V. Gheorghita, and J. Van Meerbergen, "Dataflow analysis for real-time embedded multiprocessor system design," in *Dynamic and robust streaming in and between connected consumer-electronic devices*, 2005.

[5] F. Siyoum, M. Geilen, and H. Corporaal, "Symbolic analysis of dataflow applications mapped onto shared heterogeneous resources," in *Proc. 51st annual design automation conference*, 2014.

[6] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, "A novel analytical method for worst case response time estimation of distributed embedded systems," in *Proc. 50th Annual Design Automation Conference*, 2013.

[7] L. Thiele and N. Stoimenov, "Modular performance analysis of cyclic dataflow graphs," in *Proc. 7th ACM international conference on Embedded software*, 2009.

[8] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete mathematics*, vol. 23, no. 3, pp. 309–311, 1978.

[9] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," *Computers, IEEE Transactions on*.

[10] C. D. Nielsen and M. Kishinevsky, "Performance analysis based on timing simulation," in *Proc. 31st annual Design Automation Conference*, 1994.

[11] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proc. 10th international workshop on software and compilers for embedded systems*, 2007.

[12] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *Proc. 8th ACM international conference on Hardware/software codesign and system synthesis*, 2010.

[13] A. Moonen, M. Bekooij, and J. van Meerbergen, "Timing analysis model for network based multiprocessor systems," in *Proc. 15th annual Workshop of Circuits, System and Signal Processing*, 2004.

[14] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems*, 1993.

[15] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs." in *Proc. Design Automation and Test Conference*, 2003.

[16] W. Stadje, "The busy period of the queueing system m/g/," *Journal of Applied Probability*, 1985.

[17] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Proc. 6th International Conference on Application of Concurrency to System Design*, 2006.