Combining Offsets with Precedence Constraints to Improve Temporal Analysis of Cyclic Real-Time Streaming Applications

Philip S. Kurtin* Joost P.H.M. Hausmans* Marco J.G. Bekooij*‡
philip.kurtin@utwente.nl joost.hausmans@utwente.nl marco.bekooij@nxp.com
*University of Twente, Enschede, The Netherlands ‡NXP Semiconductors, Eindhoven, The Netherlands

***Abstract*—Stream processing applications executed on multiprocessor systems usually contain cyclic data dependencies due to the presence of bounded FIFO buffers and feedback loops, as well as cyclic resource dependencies due to the usage of shared processors. In recent works it has been shown that temporal analysis of such applications can be performed by iterative fixed-point algorithms that combine dataflow and response time analysis techniques. However, these algorithms consider resource dependencies based on the assumption that tasks on shared processors are enabled simultaneously, resulting in a significant overestimation of interference between such tasks.**

This paper extends these approaches by integrating an explicit consideration of precedence constraints with a notion of offsets between tasks on shared processors, leading to a significant improvement of temporal analysis results for cyclic stream processing applications. Moreover, the addition of an iterative buffer sizing enables an improvement of temporal analysis results for acyclic applications as well.

The performance of the presented approach is evaluated in a case study using a WLAN transceiver application. It is shown that 56% higher throughput guarantees and 52% smaller end-to-end latencies can be determined compared to state-of-the-art.

I. Introduction

Real-time stream processing applications such as Software Defined Radios (SDRs) that are executed on multiprocessor systems usually require to give temporal guarantees at design time, ensuring that throughput and latency constraints can be always satisfied. In many cases, however, a temporal analysis to obtain such guarantees is not trivial, as both cyclic data dependencies and processor sharing with run-time schedulers heavily influence the temporal behavior of an analyzed application. Besides, so-called cyclic resource dependencies occur wherever resource dependencies introduced by processor sharing are opposite to the flow of data, making temporal analysis even more challenging.

It has been shown that dataflow analysis techniques are capable of obtaining temporal guarantees under such demanding circumstances [17], [19], [23]. The applicability of dataflow analysis techniques is not limited to temporal analysis, but also includes the computation of required buffer capacities [24], scheduler settings [22], a suitable task-to-processor assignment [18] and forms the basis for synchronization overhead minimization techniques such as task clustering [5] and resynchronization [8].

Especially the inherent support of cyclic data dependencies, which is enabled by the so-called the-earlier-the-better refinement relation [7], distinguishes dataflow analysis techniques from other approaches. Cyclic data dependencies regularly occur due to the presence of feedback loops. Moreover, cyclic data dependencies are also introduced by the usage of First-In-First-Out (FIFO) buffers with blocking writes for inter-task communication, i.e. buffers on which a writing task is suspended if the buffer is full. Data dependencies become cyclic for such buffers as a reading task does not only have to wait for a writing task if the buffer is empty (forward dependency), but the writing task also has to wait for the reading task if the buffer is full (backward dependency).

It is possible to assume during temporal analysis that buffer capacities are all infinite and to determine sufficiently large buffer capacities, i.e. buffer capacities that are large enough such that writing tasks never have to wait for reading tasks, afterwards. This allows to treat applications without feedback loops as acyclic, enabling the usage of accurate temporal analysis approaches that are not available for cyclic applications. Unfortunately, assuming buffer capacities as infinite during temporal analysis also prevents the exploitation of a recently discovered correlation between cyclic data dependencies and interference: If two tasks that interfere with each other, i.e. that are executed on the same processor, are connected via a cyclic data dependency then the number of interferences of one task during an execution of the other is limited by this cyclic dependency. This correlation is considered in the dataflow analysis approach from [26] and further exploited in [25] by the introduction of an iterative buffer sizing.

However, the correlation between cyclic dependencies and interference relies on the existence of cyclic data dependencies between tasks sharing a processor. If between some tasks communication is realized via FIFO buffers without blocking writes or if the capacities of some of these buffers grow large then the algorithm in [25] falls back to the rather crude response time analysis from [9] for these tasks, which basically assumes that all tasks on a shared processor are externally enabled, i.e. put in the ready queue of the scheduler after satisfaction of data dependencies, simultaneously. This can lead to a significant overestimation of interference and consequently unsatisfactory analysis results.

This paper presents a dataflow analysis approach for both cyclic and acyclic single-rate real-time stream processing applications executed on multiprocessor systems with shared processors and static priority preemptive schedulers [3]. The approach is based on the iterative fixed-point algorithm from [25] and addresses the overestimation of interference between tasks by the introduction of execution intervals. Execution intervals are a generalization of dynamic offsets, as they do not only represent bounds on external enabling times, but are defined by the minimum external enabling and maximum finish times of tasks. Furthermore, execution intervals are integrated with an explicit consideration of precedence constraints, a notion of both cyclic and acyclic data dependencies. The combination of execution intervals and precedence constraints is used to derive a more accurate, while still temporally conservative characterization of interference and consequently improved temporal guarantees for cyclic applications. Finally, the presented approach is extended with the iterative buffer sizing from [25], which enables a significant improvement of temporal guarantees for acyclic applications as well.

The remainder of this paper is structured as follows: Section II presents related work. An intuitive introduction to the presented approach is given in Section III. Section IV details the temporal analysis approach and discusses temporal conservativeness, monotonicity and convergence of the analysis flow. The approach is evaluated in a case study in Section V and Section VI presents the conclusions.

978-1-4673-8641-8/16/$31.00 ©2016 IEEE

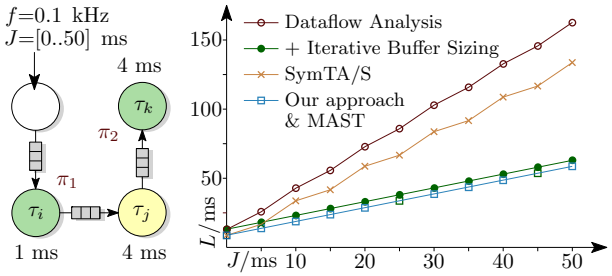

Fig. 1. Left: Task graph with variable source jitter. Right: Source jitter J vs. end-to-end latency L for different temporal analysis approaches.

II. Related Work

In this section we discuss related work. We make use of the example depicted in Figure 1, a benchmark taken from [15], to illustrate the differences between the various temporal analysis approaches. The left side of the figure shows the task graph of a streaming application, with tasks τ_i and τ_k being executed on a shared processor using a static priority preemptive scheduler and task τ_k having a higher priority (π_2) than task τ_i (π_1). The Worst-Case Execution Times (WCETs) of the tasks are written next to the tasks. The application is driven by a periodic source with the frequency $f = 0.1$ kHz and a variable jitter $J = [0..50]$ ms. Moreover, inter-task communication is realized via FIFO buffers whose optimum capacities are unknown before analysis. The right side of Figure 1 depicts the end-to-end latency L determined by different analysis approaches drawn against the source jitter J.

The approach from [9] can handle cyclic data dependencies, but determines interference between tasks solely based on a rather crude period-and-jitter characterization which assumes that all interfering tasks are externally enabled simultaneously, resulting in the curve with circles. The consideration of the effect that cyclic data dependencies bound interference in [26] also does not help since the task graph is acyclic, leading to the same curve with circles for this approach.

Only the combination with an iterative buffer sizing, which is described in [25] and also later in this paper, leads to a significant improvement of analysis results: The iterative buffer sizing makes the previously acyclic task graph cyclic, which allows to bound interference using these cycles. This technique results in the curve with dots, but requires the usage of FIFO buffers with blocking writes. Note that the differences between our approach and the aforementioned are further detailed in Section V.

The limitations of the crude period-and-jitter characterization can be addressed by the usage of offsets, which represent bounds on the external enabling times of tasks. The external enabling time of a task is thereby closely related to the so-called release time [20]: Both indicate the time at which a task iteration is put in the ready queue of its scheduler, with the difference that a release time occurs periodically, whereas the external enabling time reflects the enabling of a task iteration due to satisfaction of its data dependencies. Offsets in the context of temporal analysis were pioneered in [20]. The offsets used in this approach are static, however, limiting applicability to systems with strictly periodic schedules. This limitation is relaxed in [13] by the introduction of dynamic offsets, which combine static offsets with jitters and allow to consider systems with data-driven schedules as well.

The SymTA/S approach [11], one of the approaches discussed in [15], characterizes interference between tasks using dynamic offsets. However, the approach cannot take into account arbitrary cyclic data dependencies. Consequently, it has to assume infinite FIFO buffer capacities during analysis.

Moreover, dynamic offsets have a significant shortcoming when tasks with large jitters are involved, as it is the case in our example: The analysis has to assume for the worst case that an iteration n of task τ_i executes as late as possible (that is, with maximum jitter), whereas iteration n of task τ_k executes as early as possible (with minimum jitter). This leads to the detection of interference between iteration n of task τ_i and iteration n of task τ_k, which obviously cannot occur in the application itself. An overestimation of interference is the consequence, which is illustrated by the curve with crosses.

This problem can be addressed by replacing dynamic offsets with relative offsets, which are not defined in relation to the source, but directly in relation to interfering tasks, as it is proposed in [10]. While analysis accuracy is high for this approach, its applicability is limited to tree-shaped task graphs.

Another possibility to reduce the inaccuracies of dynamic offsets is the addition of explicit exclusions of interference due to precedence constraints. Such a method was first considered in [27] in which preempted tasks are analyzed in chains with other preempted tasks, preventing to account for the same interference multiple times. In [12] a notion of execution intervals is used in combination with precedence constraints, similar to our approach. However, precedence relations are only evaluated between tasks in the same iteration and only acyclic applications are considered.

Limiting interference using precedence constraints is also discussed in [14] and refined in [16]. The latter refinement is implemented as part of the Modeling and Analysis Suite for Real-Time Applications (MAST) [6] and is capable of an accurate characterization of interference for our example. This is illustrated by the curve with boxes, which is equal to the curve determined for our approach. However, the applicability of the technique in [16] is also limited to acyclic task graphs (in fact, even tree-shaped task graphs). Besides the obvious shortcoming that task graphs with feedback loops cannot be modeled, this can also have a negative impact on analysis accuracy for acyclic graphs.

Assume for instance that the priorities in our example are reversed, i.e. that task τ_i has a higher priority than task τ_k. The technique in [16] would conclude that an iteration n of task τ_k could potentially experience interference from all iterations of task τ_i with a higher iteration index than n. In contrast, our approach with its iterative buffer sizing would introduce additional backward dependencies from task τ_k to task τ_i that bound interference between these tasks, resulting in more accurate analysis results.

As far as we know our approach is the first to consider offsets in the context of applications with arbitrary cyclic data and resource dependencies. The resulting support for feedback loops and FIFO buffers with predefined capacities distinguishes our work from all other offset-based approaches. Moreover, our approach does not only support cyclic data dependencies natively, but uses the precedence constraints imposed by such dependencies to bound interference. To the best of our knowledge, this combination of offsets and precedence constraints enables a higher analysis accuracy for cyclic applications than achievable by any other approach.

On top of that, our analysis approach can apply an iterative buffer sizing, making even acyclic task graphs cyclic. The additionally introduced backward dependencies are used to bound interference, which can lead to a significant improvement of analysis results for acyclic applications as well.

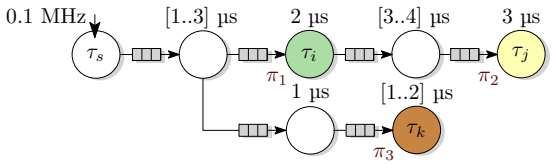

Fig. 2. Task graph example.

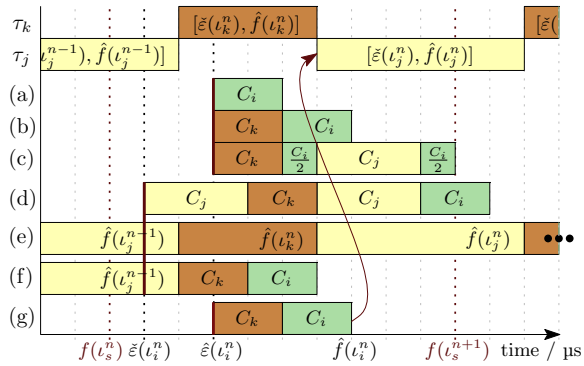

Fig. 3. Determining the maximum finish time of an iteration n of task τ_i from Figure 2.

III. Basic Idea

In this section we illustrate the basic idea of combining execution intervals with precedence constraints to bound interference between tasks on shared processors. At first, we show that there exist two different ways of conservatively bounding interference using execution intervals and that only a combination of the two leads to usable results. Then we illustrate how this bound can be combined with an explicit consideration of precedence constraints in order to determine a more accurate interference characterization.

Consider the task graph depicted in Figure 2. All tasks are triggered by the periodic source τ_s with the frequency $f = 0.1$ MHz and communicate via FIFO buffers. The Best-Case Execution Times (BCETs) and WCETs of the tasks are denoted above them. Moreover, the tasks τ_i, τ_j and τ_k are executed on a shared processor using a static priority preemptive scheduler, with task τ_i having the lowest priority π_1 and task τ_k having the highest priority π_3. The unlabeled tasks are executed on other, unshared processors and determine the external enabling and finish times of the tasks τ_i, τ_j and τ_k relative to the source. In the following we attempt to compute an as accurate as possible, yet temporally conservative (i.e. pessimistic) bound on the finish time of an iteration n of task τ_i (shorthand notation: task iteration ι_i^n). Thereby we make use of so-called execution intervals of higher priority task iterations, i.e. intervals being defined by the minimum external enabling and maximum finish times of such task iterations.

Consider the Gantt chart depicted in Figure 3. The upper part of the chart shows the execution intervals of all iterations of the higher priority tasks τ_j and τ_k that lie in the temporal proximity of iteration ι_i^n. The execution intervals are established by the minimum external enabling times $\check{\varepsilon}$ and maximum finish times $\hat{f}$ of the tasks. Initially, these are determined by the sums of BCETs and WCETs on the paths to the respective tasks, relative to the periodic executions of the source indicated by $f(\iota_s^n)$. For instance, the initial execution interval of task iteration ι_k^n is equal to $f(\iota_s^n) + [1+1, 3+1+2]$ µs $= f(\iota_s^n) + [2, 6]$ µs.

Let us first assume that iteration ι_i^n is externally enabled as late as possible, i.e. it holds that $\varepsilon(\iota_i^n) = \hat{\varepsilon}(\iota_i^n)$. Given this external enabling time, the execution intervals of the tasks τ_j and τ_k, as well as their WCETs C_j and C_k we can bound interference of the tasks in an accurate, yet temporally conservative manner, as we explain in the following.

At first, we assume that iteration ι_i^n does not experience any interference, i.e. it holds that its finish time is equal to $\hat{\varepsilon}(\iota_i^n) + C_i$ (Figure 3a). Now we see that the execution interval of iteration ι_k^n overlaps with the execution of iteration ι_i^n. Consequently, we consider interference from this overlapping iteration maximally, i.e. as the WCET of task τ_k (Figure 3b). Due to this interference the finish time of iteration ι_i^n increases, causing an additional overlap with iteration ι_j^n that we also consider maximally (Figure 3c). We apply this procedure of iteratively increasing interference until the finish time of iteration ι_i^n converges, i.e. until the point in time at which no new overlaps have to be considered. After considering interference from iteration ι_j^n no other execution intervals of higher priority tasks can overlap with the execution of iteration ι_i^n, making the finish time presented in Figure 3c an upper bound on the finish time of iteration ι_i^n for the case that it is externally enabled at $\hat{\varepsilon}(\iota_i^n)$.

Now consider that iteration ι_i^n is not externally enabled as late as possible, but as early as possible, i.e. at $\varepsilon(\iota_i^n) = \check{\varepsilon}(\iota_i^n)$. Applying the same method of iteratively adding interference we find the finish time shown in Figure 3d. As one can see, in this case the finish time of iteration ι_i^n is larger than the finish time for $\hat{\varepsilon}(\iota_i^n)$. This is due to the fact that the additional overlap with iteration ι_j^{n-1}, which is caused by the earlier external enabling, is smaller than the WCET of task τ_j.

Such an anomaly on the computed finish time bounds with respect to external enabling times earlier than $\hat{\varepsilon}(\iota_i^n)$ imposes a problem in terms of computational efficiency: If we want to find a bound on the finish time of a task iteration that is independent of when the iteration is actually externally enabled we do not only have to compute the finish time for one external enabling time, but for all external enabling times between $\check{\varepsilon}(\iota_i^n)$ and $\hat{\varepsilon}(\iota_i^n)$. This would make our proposed algorithm practically unusable.

Fortunately, there is another temporally conservative way to bound interference: A higher priority iteration cannot only interfere no more than the WCET of the corresponding task, but also no later than the maximum finish time of that iteration, i.e. no later than until the end of its execution interval. In Figure 3e this method is applied on the execution intervals of the tasks τ_j and τ_k: Starting from the external enabling time $\check{\varepsilon}(\iota_i^n)$ of iteration ι_i^n we iteratively add interference from iterations of the tasks τ_j and τ_k up to their maximum finish times. In this example, however, there is no time between the execution intervals of the higher priority task iterations. For that reason there is also no execution time left for the WCET of iteration ι_i^n and its finish time consequently converges towards infinity.

Now let us combine the two methods of bounding interference as follows: For all higher priority task iterations whose execution intervals end before or at the maximum external enabling time of iteration ι_i^n we bound interference using their finish times and for all other interfering iterations, i.e. all iterations whose maximum finish times are larger than $\hat{\varepsilon}(\iota_i^n)$, we bound interference by the corresponding WCETs. For $\varepsilon(\iota_i^n) = \check{\varepsilon}(\iota_i^n)$ the finish time depicted in Figure 3f is computed, which is apparently smaller than the finish time computed for $\varepsilon(\iota_i^n) = \hat{\varepsilon}(\iota_i^n)$. As we show later, the latter observation does not only hold for this example and not only for the case that ι_i^n is externally enabled at its minimum external enabling time, but for any external enabling time smaller than $\hat{\varepsilon}(\iota_i^n)$. For that reason we can compute an upper

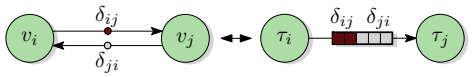

Fig. 4. One-to-one relation between dataflow model and task graph.

bound on the finish time of iteration ι_i^n by only computing an upper bound assuming that iteration ι_i^n is externally enabled at $\hat{\varepsilon}(\iota_i^n)$ and know that this bound is indeed an upper bound for any enabling time smaller or equal to $\hat{\varepsilon}(\iota_i^n)$.

Moreover, we can combine the consideration of interference based on execution intervals with an explicit consideration of precedence constraints: As it can be seen in Figure 2 task τ_i precedes task τ_j in such way that an iteration ι_i^n must be always finished before iteration ι_j^n is externally enabled. From this follows that iteration ι_j^n cannot interfere with iteration ι_i^n, independent of any overlaps with the execution interval of iteration ι_j^n. Considering this relation in the computation of maximum interference then just results in the finish time of iteration ι_i^n presented in Figure 3g. In the following section we show how to use dataflow modeling to derive a consistent way of considering precedence constraints between tasks in the finish time computation, resulting in more accurate analysis results than achievable by only considering execution intervals.

Finally, note that the determined difference between maximum external enabling and maximum finish time of task iteration ι_i^n is larger than the WCET of task τ_i. This results in an additional delay of task iteration ι_j^n that has to be accounted for in a larger execution interval. Larger execution intervals, however, can lead to even more interference that must be considered. Therefore we have to compute execution intervals and finish times alternately until convergence is achieved. This makes our algorithm iterative.

IV. Temporal Analysis

Our temporal analysis and buffer sizing approach is presented in this section. Section IV-A describes the analysis model and Section IV-B the iterative analysis flow. Section IV-C presents algorithms that are used to determine periodic execution intervals of task iterations, taking into account delays due to data dependencies. In Section IV-D these execution intervals, as well as precedence constraints, are employed to bound finish times of task iterations not only considering data dependencies, but also considering resource dependencies due to processor sharing. Finally, Section IV-E describes the iterative buffer sizing.

In the remainder of this paper we refer to upper (lower) bounds on external enabling times of tasks as maximum external enabling times $\hat{\varepsilon}$ (minimum external enabling times $\check{\varepsilon}$). Analogously, we refer to upper (lower) bounds on finish times as maximum finish times $\hat{f}$ (minimum finish times $\check{f}$). Moreover, we call an iteration n of a task τ_i task iteration ι_i^n and the source period of that task P_i.

A. Analysis Model

We make use of Homogeneous Synchronous Dataflow (HSDF) graphs to calculate lower bounds on the best-case and upper bounds on the worst-case schedule of an analyzed application. These bounds on schedules are used for the verification of temporal constraints, the derivation of execution intervals and a calculation of sufficiently large buffer capacities.

An HSDF graph is a directed graph $G = (V, E, \delta, \rho)$ that consists of a set of actors V and a set of directed edges E connecting these actors. An actor $v_i \in V$ communicates with other actors by producing tokens on and consuming tokens from edges, which represent unbounded queues. An edge $e_{ij} = (v_i, v_j) \in E$ initially contains $\delta(e_{ij})$ tokens. An actor v_i is enabled to fire if at least one token is available on each of its incoming edges. Furthermore, the firing duration ρ_i specifies the difference between the start and finish times of a firing of an actor v_i. An actor consumes one token from each of its incoming edges at the start of a firing and produces one token on each of its outgoing edges when a firing finishes.

With our approach we analyze applications $\mathcal{A}$ that can be described by one or more task graphs $\mathcal{T} \in \mathcal{A}$. We specify a task graph $\mathcal{T}$ as a weakly connected directed graph, with its vertices $\tau_i \in \mathcal{T}$ representing tasks and its directed edges representing FIFO buffers. Our analysis requires BCETs and WCETs of all tasks that hold independent of schedules. The underlying hardware must support obtaining these times, as does for instance the Starburst architecture [4]. Thereby it can be assumed that all tasks are executed in isolation, since processor sharing is analyzed by our algorithm. Communication time, however, has to be included in the WCETs of tasks.

Furthermore, we require that a task is only externally enabled, i.e. put in the ready queue of the scheduler, if data is available in all its input buffers and free space in all its output buffers (data-driven scheduling).

Each task graph is single-rate and has a single strictly periodic source τ_s that directly or indirectly externally enables all other tasks of the task graph by writing data to input buffers of tasks. Without loss of generality we require that no task is externally enabled before the first execution of the source τ_s, which is the case if for each task at least one input buffer is initially empty. In the following all times of a task graph are defined relative to that first source execution.

Writing data to an output buffer can be implemented with the following three steps. At first, it is verified whether an output buffer location is not locked by a reading task. If this is not the case, the location is locked by the writing task (acquisition of space). This is followed by the actual write operation to the locked buffer location (data write) and finalized by unlocking the buffer location, making it available to reading tasks again (release of data). Analogously, reading data from an input buffer can be characterized by an acquisition of data, a data read and a release of space. FIFO behavior can then be implemented by simply traversing buffer locations on both read and write operations in sequential order, with a wrap-around after the last location.

As depicted in Figure 4 we model each task of a task graph as a single HSDF actor. Such a one-to-one relation between tasks and actors can be maintained if a scheduler performs all acquisition operations of both data and space at the beginning and all release operations at the end of each task execution.

Exchanging data between tasks over a FIFO buffer can then be modeled by a directed cycle in an HSDF graph, as depicted in Figure 4, with the number of initial tokens δ_{ij} on the edge from actor v_i to actor v_j being equal to the number of initially full containers in the corresponding FIFO buffer and the number of initial tokens δ_{ji} on the edge from actor v_j to actor v_i being equal to the number of initially free containers. The consumption of a token by actor v_i then corresponds to an acquisition of space, whereas a token production by that actor corresponds to a release of data. Analogously, the consumption of a token by actor v_j corresponds to an acquisition of data and the production of a token by actor v_j to a release of space.

In the following we derive HSDF graphs from task graphs to compute minimum and maximum start times of actors. These start times are then used to compute bounds on the external enabling and finish times of the corresponding tasks.

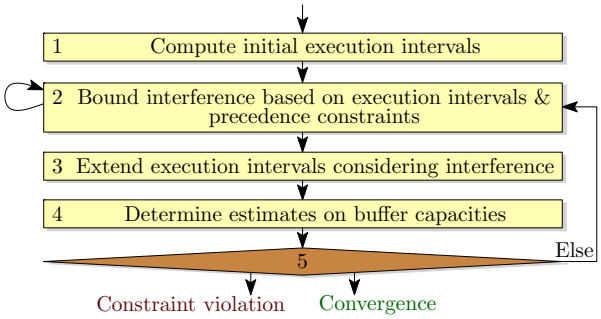

Fig. 5. Overview of the analysis flow.

B. Iterative Analysis Flow

Figure 5 depicts the flow of our temporal analysis approach. We use this analysis flow for the verification of temporal constraints and for the computation of buffer capacities that allow for a satisfaction of these constraints.

Input to our analysis flow are an application consisting of one or more task graphs, a fixed task-to-processor mapping, a specification of scheduler settings and a set of temporal constraints. Moreover, if buffer capacities are determined with our analysis flow then upper bounds on these capacities can be specified.

In step 1 initial execution intervals are computed. Based on the correspondence depicted in Figure 4, two HSDF models are derived that reflect the best-case and worst-case behavior of the analyzed application. The firing durations of the actors in the best-case model are set to the BCETs of the corresponding tasks, which enables the computation of periodic lower bounds on the external enabling times of tasks with this model. Analogously, the firing durations of actors in the worst-case model are set to the WCETs of the corresponding tasks. As we show in the next section, this model can be used to derive periodic upper bounds on the external enabling times of tasks, i.e. periodic upper bounds on the times at which task iterations are put in the ready queues of their schedulers. By adding the WCETs of tasks to the determined maximum external enabling times a first periodic upper bound on the finish times of tasks is derived. The minimum external enabling times and the maximum finish times then just form the sought initial execution intervals.

Note that at this stage the maximum finish times only account for delays due to data dependencies, but not for delays due to resource dependencies. This is amended in step 2 of the analysis flow, which estimates interference between tasks executed on shared processors. We consider a combination of precedence constraints and the execution intervals determined in step 1 of the flow to determine accurate estimates on interference. Starting from the previously determined maximum external enabling times we iteratively add interference, according to Section III, until upper bounds on the finish times of tasks are determined. These bounds are now conservative with respect to both data and resource dependencies.

However, the increased maximum finish times from step 2 can also result in additional delays of tasks due to data dependencies. Consequently, in step 3 we have to recompute the maximum finish times derived from the worst-case model in step 1, with the firing durations of actors set to the differences between previously determined maximum external enabling times and the maximum finish times computed in step 2.

The maximum finish times computed in step 3, however, can result in enlarged execution intervals, which can potentially lead to even more interference than considered in step 2. Therefore step 2 has to be repeated, taking the extended execution intervals into account. This is the reason why our analysis flow is iterative.

To combine our temporal analysis with an iterative buffer sizing we determine estimates on buffer capacities in step 4 of the flow. The capacities are computed based on the schedules and maximum finish times computed in step 3. In turn, the estimates on buffer capacities define additional precedence constraints that are used to bound interference in step 2.

Finally, the schedules are checked against temporal constraints in step 5 and it is verified whether all maximum external enabling times and buffer capacities have converged, i.e. have not changed since the previous iteration of the algorithm. If a constraint is violated the algorithm stops. Otherwise, depending on whether all maximum external enabling times and buffer capacities have converged the algorithm either finishes or repeats the steps 2 to 5 until either convergence is achieved or constraints are violated.

In the following we ensure that both maximum external enabling times and buffer capacities increase monotonically throughout iterations of the analysis flow. Moreover, it can be seen that both maximum external enabling times and buffer capacities increase with a minimum step size, which ensures that no enabling time nor buffer capacity can converge towards a certain value indefinitely. If additionally upper bounds are specified on all buffer capacities or if maximum latencies are defined it follows that all maximum external enabling times and buffer capacities are limited from above. This combination of monotonicity, minimum step sizes and limits ensures that the analysis flow terminates, as it is detailed in [25].

As the other algorithms discussed in Section II our analysis flow is iterative, both due to the mutual dependency between interference and execution intervals (outer loop) and within the interference computations (inner loop). The computational complexity of our flow is therefore in the worst-case non-polynomial. Nevertheless, the run-times are usually small enough for an off-line algorithm, as shown in Section V.

C. Determining Execution Intervals

In this section we present our method to compute execution intervals in steps 1 and 3 of the analysis flow. The execution intervals are formed by periodic lower bounds on the external enabling times and periodic upper bounds on the finish times of tasks. Following the method presented in [9] we compute these bounds using dataflow models reflecting the best-case and worst-case behavior of an analyzed application.

To determine lower bounds on external enabling times of tasks in step 1 of the analysis flow we consider a best-case model in which each task is modeled as a dataflow actor, according to Figure 4. The firing durations $\check{\rho}$ of actors in this best-case model are thereby set to the BCETs of the corresponding tasks. Furthermore, cyclic data dependencies with limited numbers of tokens can delay periodic start times of actors, while it can occur that the corresponding tasks do not always experience the same delays. Consequently, we only consider edges without initial tokens in the best-case model. Taking the latter restriction into account, it has been shown in [9] that the following Linear Program (LP) can be used to determine start times of actors in the best-case model:

$$\begin{aligned} \text{Minimize } & \sum_{v_i \in V} \check{s}_i \\ \text{Subject to: } & \check{s}_s = 0,\ \forall_{e_{ij} \in E'}:\ \check{s}_j - \check{s}_i \geq \check{\rho}_i \\ \text{with } & E' = \{e \,|\, e \in E \ \wedge\ \delta(e) = 0\} \end{aligned}$$

These start times define a periodic schedule for the actors in the best-case model, i.e. it holds that the start time of an actor v_i in iteration n is equal to $\check{s}_i + n \cdot P_i$. By setting the start time of the source actor v_s to $\check{s}_s = 0$ it holds that all start times are computed relative to the first enabling of the source actor. As the external enabling time of a task is also defined relative to the first execution of its strictly periodic source it follows that we can use the start time $\check{s}_i$ of an actor v_i in the best-case model to determine a periodic lower bound on the external enabling times $\varepsilon(\iota_i^n)$ of the corresponding task τ_i, i.e.:

$$\forall_{n \geq 0} \colon \ \check{\varepsilon}(\iota_i^n) = \check{s}_i + n \cdot P_i \leq \varepsilon(\iota_i^n) \tag{1}$$

Moreover, we define a worst-case model that is used to compute upper bounds on the external enabling times of tasks in steps 1 and 3 of the analysis flow. According to Figure 4 we consider buffer capacities as cyclic data dependencies in the worst-case model. The numbers of tokens for buffer capacities determined with our analysis flow are set to the upper bounds specified as input to the flow. This is required as using estimates on buffer capacities instead of upper bounds could lead to both larger maximum finish times in step 2 and smaller maximum external enabling times in step 3, which would consequently break monotonicity of the analysis flow.

In step 1 of the analysis flow we set the firing durations $\hat{\rho}$ of actors in the worst-case model to the WCETs of the corresponding tasks. In step 3, however, the firing durations are set to the differences between the maximum external enabling and the maximum finish times of tasks, with the first either computed in step 1 or, if applicable, in the previous iteration of step 3 and the latter computed in step 2 of the analysis flow. According to [9] the start times of actors in the worst-case model can be computed by solving the following LP:

$$\text{Minimize} \sum_{v_i \in V} \hat{s}_i \tag{2}$$

$$\text{Subject to: } \hat{s}_s = 0, \ \forall_{e_{ij} \in E} \colon \ \hat{s}_j - \hat{s}_i \geq \hat{\rho}_i - \delta(e_{ij}) \cdot P_i$$

Also note that self-edges with one token are usually used in dataflow models to capture that a task cannot be enabled before its previous execution is finished. However, we are not making use of the actual enabling times of tasks, but of external enabling times. Hence we omit self-edges in the worst-case model, which has the additional advantage that the differences between the maximum external enabling and maximum finish times of tasks are allowed to be larger than source periods. The data dependencies between tasks are nevertheless exactly captured by the edges between the corresponding actors. This allows to bound the external enabling time $\varepsilon(\iota_i^n)$ of a task iteration ι_i^n from above as follows:

$$\forall_{n \geq 0} \colon \ \varepsilon(\iota_i^n) \leq \hat{\varepsilon}(\iota_i^n) = \hat{s}_i + n \cdot P_i \tag{3}$$

The sum of this periodic upper bound on the external enabling times of task iterations ι_i^n and the firing duration of the actor corresponding to task τ_i defines an upper bound on the finish times $f(\iota_i^n)$ of task iterations ι_i^n, i.e.:

$$\forall_{n \geq 0} \colon \ f(\iota_i^n) \leq \hat{f}(\iota_i^n) = \hat{s}_i + \hat{\rho}_i + n \cdot P_i \tag{4}$$

With Equations 1 and 4 we can finally define the sought periodic execution intervals:

Definition 1: The execution interval $\mathcal{I}(\iota_i^n)$ of a task iteration ι_i^n is defined as the interval between its minimum external enabling time and its maximum finish time, i.e.:

$$\mathcal{I}(\iota_i^n) = [\check{\varepsilon}(\iota_i^n), \hat{f}(\iota_i^n)] = [\check{s}_i + n \cdot P_i, \hat{s}_i + \hat{\rho}_i + n \cdot P_i]$$

D. Bounding Interference

In this section we present the derivation of periodic and temporally conservative upper bounds on the finish times of tasks with respect to resource dependencies. At first, we bound the finish time of a single task iteration ι_i^n from above, by considering interference from higher priority tasks executed on the same processor as task τ_i in a temporally conservative manner. This means that external enabling and execution times of higher priority tasks are considered in such way that interference on task τ_i is maximized. To bound interference of higher priority tasks we make use of WCETs, the execution intervals defined in the previous section, as well as precedence constraints. Then we extend the derived single-iteration bound to a periodic bound that holds for any task iteration. Finally, we use these periodic upper bounds on finish times to derive the firing durations required in step 3 of the analysis flow.

The outline of the remainder of this section is as follows: In Section IV-D1 it is pointed out that it is not sufficient to consider all interference of higher priority tasks that can occur between the external enabling time of an iteration ι_i^n and its finish time, since earlier iterations $\iota_i^{n-1}, \iota_i^{n-2}, \ldots$ can also delay the execution of iteration ι_i^n. Subsequently, we present a temporally conservative method of considering such self-interference. In Section IV-D2 it is explained how execution intervals can be used to bound interference and in Section IV-D3 how the accuracy of this bound on interference can be improved by an explicit consideration of precedence constraints. In Section IV-D4 a distinction between interference from tasks belonging to the same and other task graphs is made, which allows a simplification of the derived interference bound. The transformation of a bound on the finish time of a single iteration ι_i^n to a periodic bound valid for any iteration of task τ_i is explained in Section IV-D5, as is the subsequent computation of firing durations for the worst-case model.

1) Handling Self-Interference: It is important to note that the finish time of a task iteration cannot only be delayed due to higher priority tasks that interfere with this iteration, but also due to previous iterations of the same task which are not finished before the iteration is externally enabled. For that reason it is not sufficient to compute an upper bound on the finish time of an iteration ι_i^n with respect to its own external enabling time only. Instead, we have to determine upper bounds on the finish time of an iteration ι_i^n with respect to its own external enabling time and the external enabling times of all preceding iterations $\iota_i^{n-1}, \iota_i^{n-2}, \ldots$. Computing the maximum of all these bounds then results in an upper bound on the finish time of iteration ι_i^n that is temporally conservative regarding both interference of higher priority tasks and self-interference. To obtain such a bound we make use of so-called maximum busy periods:

Definition 2: The maximum busy period $w_i(\varepsilon(\iota_i^{n-q+1}), q)$ defines an upper bound on the time between the external enabling of an iteration ι_i^{n-q+1} and the finish of an iteration ι_i^n under the following two assumptions:

- Iteration ι_i^{n-q+1} is not delayed by its preceding iteration, i.e. it holds that $f(\iota_i^{n-q}) \leq \varepsilon(\iota_i^{n-q+1})$.
- All q iterations $\iota_i^{n-q+1} \ldots \iota_i^n$ are in consecutive execution, i.e. it holds for all iterations ι_i^m and ι_i^{m+1} with $m \in \{n-q+1 \ldots n-1\}$ that $f(\iota_i^m) \geq \varepsilon(\iota_i^{m+1})$.

Note that this definition of a maximum busy period is in line with the definition given in [13], with the difference that in [13] a maximum busy period begins with the critical instant, i.e. the

earliest point in time before or at the external enabling of a task iteration ι_i^n after which in the worst-case no lower priority task of task τ_i can execute until the end of the maximum busy period. In contrast, our maximum busy period begins at the external enabling time of a task iteration ι_i^n, which is potentially later than the critical instant. This is enabled by the usage of execution intervals instead of external enabling intervals, that are used in [13] to characterize interference.

If for a maximum busy period $w_i(\varepsilon(\iota_i^{n-q+1}), q)$ both assumptions hold it follows by definition that $\varepsilon(\iota_i^{n-q+1}) + w_i(\varepsilon(\iota_i^{n-q+1}), q)$ is a temporally conservative upper bound on the finish time of a task iteration ι_i^n. It can be seen that for each task iteration ι_i^n there must exist at least one $q \geq 1$ for which both assumptions hold. Moreover, it holds for the external enabling time of an iteration ι_i^{n-q+1}:

$$\varepsilon(\iota_i^{n-q+1}) \in \mathcal{E}(\iota_i^{n-q+1}) = [\check{\varepsilon}(\iota_i^{n-q+1}), \hat{\varepsilon}(\iota_i^{n-q+1})]$$

Taking this range of external enabling times into account, it can be seen that a temporally conservative bound on the finish time of iteration ι_i^n with respect to both interference of higher priority tasks and self-interference can be computed as follows (with ε a shorthand notation for $\varepsilon(\iota_i^{n-q+1})$):

$$\hat{f}(\iota_i^n) = \max_{q \geq 1}\Big(\max_{\varepsilon \in \mathcal{E}(\iota_i^{n-q+1})}(\varepsilon + w_i(\varepsilon, q))\Big) \tag{5}$$

Now that we have established a relation between maximum busy periods and maximum finish times we focus on the derivation of an accurate, yet temporally conservative function to compute maximum busy periods for any external enabling times and numbers of consecutive executions. After deriving such a function we show that Equation 5 can be simplified in such way that it becomes computable, even though the number of possible external enabling times within $\mathcal{E}(\iota_i^{n-q+1})$ may be very large in general.

2) Bounding Interference using Execution Intervals: A maximum busy period $w_i(\varepsilon(\iota_i^{n-q+1}), q)$ has to be temporally conservative. This means that its value must be large enough to accommodate both the WCETs of q consecutive executions of task τ_i, as well as all interference of higher priority tasks that can occur between the external enabling time of the first of the q consecutive executions, $\varepsilon(\iota_i^{n-q+1})$, and the finish time of the last, $\varepsilon(\iota_i^{n-q+1}) + w_i(\varepsilon(\iota_i^{n-q+1}), q)$. We account for the execution time of task τ_i by simply adding q times its WCET C_i to the maximum busy period. To account for interference of higher priority tasks $\tau_j \in hp(i)$ we make use of the previously determined execution intervals. Note that in the following we use the notation $\mathcal{I}(\iota_j^m) \in w_i(\varepsilon(\iota_i^{n-q+1}), q)$ to indicate that a task iteration ι_j^m is considered as interference in $w_i(\varepsilon(\iota_i^{n-q+1}), q)$.

As execution intervals are defined by the temporally conservative minimum external enabling and maximum finish times of task iterations it can be seen that only task iterations can interfere with any of the q iterations of task τ_i whose execution intervals overlap with the maximum busy period. Therefore we only have to account for higher priority task iterations that adhere to the following two constraints:

$$\begin{aligned}&\mathcal{I}(\iota_j^m) \in w_i(\varepsilon(\iota_i^{n-q+1}), q)\\ &\Rightarrow \hat{f}(\iota_j^m) > \varepsilon(\iota_i^{n-q+1}) \wedge\\ &\quad \check{\varepsilon}(\iota_j^m) < \varepsilon(\iota_i^{n-q+1}) + w_i(\varepsilon(\iota_i^{n-q+1}), q)\end{aligned}$$

In Section III it is discussed that there are two ways of conservatively modeling interference of higher priority tasks using execution intervals. In the following we consequently divide all iterations ι_j^m that can interfere with $w_i(\varepsilon(\iota_i^{n-q+1}), q)$ into two groups: Interference from task iterations whose maximum finish times are smaller or equal to $\hat{\varepsilon}(\iota_i^{n-q+1})$ is considered via maximum finish times, whereas interference from all other iterations is considered via the WCETs of the corresponding tasks. Using this distinction we can divide the busy period $w_i(\varepsilon(\iota_i^{n-q+1}), q)$ into two parts, the first only considering interference from the first group and the second only considering interference from the second group:

$$w_i(\varepsilon(\iota_i^{n-q+1}), q) = w_i^*(\varepsilon(\iota_i^{n-q+1})) + w_i'(q) \tag{6}$$

For the iterations in the first group it holds by definition that their maximum finish times are all smaller or equal to the maximum external enabling time of iteration ι_i^{n-q+1}, i.e.:

$$\hat{f}(\iota_j^m) \leq \hat{\varepsilon}(\iota_i^{n-q+1})$$

Therefore none of these iterations can interfere with any of the q iterations of task τ_i after $\hat{\varepsilon}(\iota_i^{n-q+1})$. If we additionally substitute $\hat{\varepsilon}(\iota_i^{n-q+1})$ using Equation 3 we can bound interference from these iterations from above with the following function:

$$\begin{aligned}w_i^*(\varepsilon(\iota_i^{n-q+1})) &= \hat{\varepsilon}(\iota_i^{n-q+1}) - \varepsilon(\iota_i^{n-q+1})\\ &= \hat{s}_i + (n-q+1)\cdot P_i - \varepsilon(\iota_i^{n-q+1})\end{aligned} \tag{7}$$

Note that by using $w_i^*(\varepsilon(\iota_i^{n-q+1}))$ to bound interference we implicitly assume that the q iterations of task τ_i, as well as any higher priority task iterations with a finish time larger than $\hat{\varepsilon}(\iota_i^{n-q+1})$, do not execute before $\hat{\varepsilon}(\iota_i^{n-q+1})$. For that reason we have to define the function $w_i'(q)$ in such way that it takes the WCETs of q iterations of task τ_i into account, as well as all interference of higher priority task iterations whose maximum finish times are larger than the maximum external enabling time of iteration ι_i^{n-q+1}. For the higher priority task iterations that must be considered in $w_i'(q)$ it follows:

$$\begin{aligned}\mathcal{I}(\iota_j^m) \in w_i'(q) \Rightarrow\ &\hat{f}(\iota_j^m) > \hat{\varepsilon}(\iota_i^{n-q+1}) \wedge\\ &\check{\varepsilon}(\iota_j^m) < \hat{\varepsilon}(\iota_i^{n-q+1}) + w_i'(q)\end{aligned}$$

Replacing the minimum and maximum external enabling times and the maximum finish times with the periodic bounds derived in Section IV-C results in the following:

$$\begin{aligned}&\mathcal{I}(\iota_j^m) \in w_i'(q)\\ &\Rightarrow \quad \hat{s}_j + \hat{\rho}_j + m\cdot P_j > \hat{s}_i + (n-q+1)\cdot P_i\\ &\qquad \wedge\ \check{s}_j + m\cdot P_j < \hat{s}_i + w_i'(q) + (n-q+1)\cdot P_i\\ &\Leftrightarrow \quad \left\lfloor \frac{\hat{s}_i + (n-q+1)\cdot P_i - \hat{s}_j - \hat{\rho}_j}{P_j} \right\rfloor < m\\ &\qquad < \left\lceil \frac{\hat{s}_i + w_i'(q) + (n-q+1)\cdot P_i - \check{s}_j}{P_j} \right\rceil\end{aligned} \tag{8}$$

Note that the last equivalence holds because m must be integer.

Using Equation 8 we can summarize all iterations of a higher priority task $\tau_j \in hp(i)$ that have to be considered in $w_i'(q)$ due to their execution intervals in the so-called execution interval set:

$$\begin{aligned}E_{\tau_j \to w_i'(q)} = \{\iota_j^m \mid &\left\lfloor \frac{\hat{s}_i + (n-q+1)\cdot P_i - \hat{s}_j - \hat{\rho}_j}{P_j} \right\rfloor < m\\ &< \left\lceil \frac{\hat{s}_i + w_i'(q) + (n-q+1)\cdot P_i - \check{s}_j}{P_j} \right\rceil\}\end{aligned}$$

Before using such sets to derive a temporally conservative function $w'_i(q)$ we introduce the notion of precedence constraints in the context of interference in the next section, which enables the computation of more accurate bounds on interference than by only considering execution intervals.

3) Tightening Interference Bounds using Precedence Constraints: In this section we present a bound on interference that is based on precedence constraints. Thereby we make use of so-called precedence sets. Due to space limitations we only derive these sets intuitively, for a formal derivation of precedence sets please refer to [26].

Reconsider the task graph depicted in Figure 4, which contains a producing task τ_i that is connected to a consuming task τ_j via a FIFO buffer of the capacity $\delta_{ij} + \delta_{ji}$. We further assume that both tasks execute on the same processor and that task τ_j has a higher priority than task τ_i. Based on this we derive all iterations of task τ_j that can occur during one iteration n of task τ_i. We do this derivation in the corresponding HSDF model and use the notation ι_i^n not only for task iterations, but also for the corresponding firings of actors in the model.

According to the semantics of HSDF graphs, actor v_j can fire δ_{ij} times before it must be enabled by a completed firing of actor v_i. From this follows that all firings ι_j^m with $m \geq n+\delta_{ij}$ cannot be enabled before firing ι_i^n finishes. Moreover, it also holds that actor v_i can fire δ_{ji} times before it must be enabled by a completed firing of actor v_j. From this follows that all firings ι_j^m with $m + \delta_{ji} \leq n$ must be finished before firing ι_i^n is enabled.

Negating both constraints gives us the firings ι_j^m that can occur during a firing ι_i^n, despite the precedence constraints between the two:

$$m < n + \delta_{ij} \ \wedge \ m + \delta_{ji} > n \ \Leftrightarrow \ n - \delta_{ji} < m < n + \delta_{ij}$$

As derived in [26] we can generalize this observation from single edges to paths of edges: We define $\mathcal{P}_{ij}$ as the set of all directed paths of edges from an actor v_i to an actor v_j and $\delta(\mathcal{P}_{ij})$ as the minimum number of tokens on any path in $\mathcal{P}_{ij}$, with $\delta(\mathcal{P}_{ij}) = \infty$ if $\mathcal{P}_{ij} = \emptyset$. According to [26] we can then capture all iterations ι_j^m that can interfere with an iteration ι_i^n despite precedence constraints between the corresponding tasks in the so-called precedence set:

$$P_{\tau_j \to \iota_i^n} = \{\iota_j^m \mid n - \delta(\mathcal{P}_{ji}) < m < n + \delta(\mathcal{P}_{ij})\}$$

Note that adding an edge with an infinite number of initial tokens to an HSDF graph does not affect the start times of any actors. This relation is used in the definition of $\delta(\mathcal{P}_{ij})$ for $\mathcal{P}_{ij} = \emptyset$ to allow for a temporally conservative consideration of actors without directed paths of edges between them. In [26] it is shown that $\delta(\mathcal{P}_{ji})$ and $\delta(\mathcal{P}_{ij})$ can be computed efficiently using the Floyd-Warshall algorithm.

To bound interference on $w'_i(q)$ we do not only require all iterations ι_j^m that can interfere with one iteration ι_i^n, but that can interfere with all q consecutive iterations of task τ_i, with iteration ι_i^{n-q+1} being the first. A set containing all these interfering iterations can be obtained by taking the union of the precedence sets of the iterations $\iota_i^{n-q+1} \dots \iota_i^n$:

$$\begin{aligned} P_{\tau_j \to w'_i(q)} &= \bigcup_{n'=n-q+1}^{n} P_{\tau_j \to \iota_i^{n'}} \\ &= \{\iota_j^m \mid n - q + 1 - \delta(\mathcal{P}_{ji}) < m < n + \delta(\mathcal{P}_{ij})\} \end{aligned}$$

Both $E_{\tau_j \to w'_i(q)}$ and $P_{\tau_j \to w'_i(q)}$ are conservative upper bounds on all iterations of a task τ_j that can interfere with $w'_i(q)$. To compute a more accurate bound on interference we can therefore simply draw the intersection of the two sets. Before we do that, however, let us first investigate how the lower bounds of both sets relate to each other.

On the one hand, if there is no directed path from an actor v_j to an actor v_i in the corresponding dataflow model then $\delta(\mathcal{P}_{ji})$ is infinite, making the lower bound of $E_{\tau_j \to w'_i(q)}$ always larger than the lower bound of $P_{\tau_j \to w'_i(q)}$. The lower bound of $E_{\tau_j \to w'_i(q)}$ therefore becomes the dominant bound when the intersection between the two sets is taken. On the other hand, if there is at least one directed path $p \in \mathcal{P}_{ji}$ then it holds that the periods P_i, P_j and the periods of all other tasks on path p must be equal, since the tasks must all belong to the same task graph. By recursively substituting the inequalities from the worst-case LP in Equation 2 that are defined for the edges on a path p it follows:

$$\begin{aligned} \forall_{p \in \mathcal{P}_{ji}}: \ \hat{s}_i &\geq \hat{s}_j + \sum_{e_{ab} \in p} (\hat{\rho}_a - \delta(e_{ab}) \cdot P_j) \\ &\geq \hat{s}_j + \hat{\rho}_j - \sum_{e_{ab} \in p} \delta(e_{ab}) \cdot P_j \end{aligned}$$

As this inequality holds for any path from actor v_j to actor v_i it must also hold for the path p^* with the minimum number of tokens $\delta(\mathcal{P}_{ji}) = \sum_{e_{ab} \in p^*} \delta(e_{ab})$. From this follows:

$$\hat{s}_i \geq \hat{s}_j + \hat{\rho}_j - \delta(\mathcal{P}_{ji}) \cdot P_j \ \Leftrightarrow \ -\delta(\mathcal{P}_{ji}) \leq \frac{\hat{s}_i - \hat{s}_j - \hat{\rho}_j}{P_j}$$

$$\Leftrightarrow n - q + 1 - \delta(\mathcal{P}_{ji}) \ \leq \frac{\hat{s}_i + (n-q+1) \cdot P_i - \hat{s}_j - \hat{\rho}_j}{P_j}$$

$$\Leftrightarrow n - q + 1 - \delta(\mathcal{P}_{ji}) \ \leq \left\lfloor \frac{\hat{s}_i + (n-q+1) \cdot P_i - \hat{s}_j - \hat{\rho}_j}{P_j} \right\rfloor$$

Note that the last equivalence holds because the left hand side of the inequality is integer.

This lets us conclude that the lower bound of $E_{\tau_j \to w'_i(q)}$ is always larger than the lower bound of $P_{\tau_j \to w'_i(q)}$, independent of whether there is a path from actor v_j to actor v_i or not. Taking this observation into account by ignoring the lower bound of $P_{\tau_j \to w'_i(q)}$ we can draw the intersection between the two sets in the so-called interference set:

$$\begin{aligned} I_{\tau_j \to w'_i(q)} &= E_{\tau_j \to w'_i(q)} \cap P_{\tau_j \to w'_i(q)} \\ &= \{\iota_j^m \mid \left\lfloor \frac{\hat{s}_i + (n-q+1) \cdot P_i - \hat{s}_j - \hat{\rho}_j}{P_j} \right\rfloor < m < \\ &\quad \min(\left\lceil \frac{\hat{s}_i + w'_i(q) + (n-q+1) \cdot P_i - \check{s}_j}{P_j} \right\rceil, n + \delta(\mathcal{P}_{ij}))\} \end{aligned}$$

Finally, we are not interested in the particular iterations of tasks $\tau_j \in hp(i)$ that can interfere with $w'_i(q)$, but only in the number of iterations. For that reason we define a function $\gamma_{\tau_j \to w'_i(q)}$ returning the maximum number of iterations of a task τ_j that can interfere with $w'_i(q)$. Such a function can be determined by computing the number of elements in $I_{\tau_j \to w'_i(q)}$ (with $-\lfloor a \rfloor = \lceil -a \rceil$):

$$\begin{aligned} \gamma_{\tau_j \to w'_i(q)} &= |I_{\tau_j \to w'_i(q)}| \\ &= \min(\left\lceil \frac{\hat{s}_i + w'_i(q) + (n-q+1) \cdot P_i - \check{s}_j}{P_j} \right\rceil, n + \delta(\mathcal{P}_{ij})) \\ &\quad + \left\lceil \frac{\hat{s}_j + \hat{\rho}_j - \hat{s}_i - (n-q+1) \cdot P_i}{P_j} \right\rceil - 1 \end{aligned}$$

4) Differing between Interference from the same and other Task Graphs: We differ between the two cases that a task τ_i and a higher priority task $\tau_j \in hp(i)$ belong to the same task graph or not. This additional information can be used to simplify the function $\gamma_{\tau_j \to w'_i(q)}$. With the shorthand notation $\mathcal{T}_i$ for the task graph of a task τ_i we can rewrite $\gamma_{\tau_j \to w'_i(q)}$ as follows:

$$\gamma_{\tau_j \to w'_i(q)} = \begin{cases} \gamma^*_{\tau_j \to w'_i(q)} & , \ \mathcal{T}_j = \mathcal{T}_i \\ \gamma'_{\tau_j \to w'_i(q)} & , \text{ else} \end{cases}$$

Let us first consider the case that $\mathcal{T}_j = \mathcal{T}_i$. As both tasks belong to the same task graph they must also have the same period, i.e. $P_j = P_i$. Hence it holds that the summand $\frac{(n-q+1)\cdot P_i}{P_j} = n - q + 1$ is integer and we can draw it outside of both ceiling functions. From this follows:

$$\begin{aligned} \gamma^*_{\tau_j \to w'_i(q)} = \quad & \min(\left\lceil \frac{\hat{s}_i + w'_i(q) - \check{s}_j}{P_j} \right\rceil, \delta(\mathcal{P}_{ij}) + q - 1) \\ & + \left\lceil \frac{\hat{s}_j + \hat{\rho}_j - \hat{s}_i}{P_j} \right\rceil - 1 \end{aligned}$$

Note that by this simplification $\gamma^*_{\tau_j \to w'_i(q)}$ becomes independent of a specific iteration ι^n_i. It is therefore a valid upper bound on the number of interfering iterations of a task τ_j for any $w'_i(q)$, no matter for which q particular iterations of task τ_i it is determined.

For tasks τ_j and τ_i belonging to different task graphs $\delta(\mathcal{P}_{ij})$ is infinite. Therefore only the execution interval set has to be considered. However, the external enabling times of tasks belonging to different task graphs are generally uncorrelated, i.e. it is unknown how $\check{s}_j$ and $\hat{s}_j$ on the one hand and $\hat{s}_i$ on the other hand relate to each other. In the worst-case we have to assume that these times are correlated in such way that the inequality $\lceil a \rceil + \lceil b \rceil - 1 \leq \lceil a + b \rceil$ becomes an equality when applied on $\gamma_{\tau_j \to w'_i(q)}$. From this follows:

$$\gamma'_{\tau_j \to w'_i(q)} = \left\lceil \frac{\hat{s}_j + \hat{\rho}_j - \check{s}_j + w'_i(q)}{P_j} \right\rceil$$

As one can see it holds that $\gamma'_{\tau_j \to w'_i(q)}$ is also independent of a particular iteration ι^n_i.

5) From a Single Iteration Bound to a Periodic Bound: After having derived the function $\gamma_{\tau_j \to w'_i(q)}$ that returns the maximum number of interfering iterations of a task τ_j during $w'_i(q)$ we can now also determine the function $w'_i(q)$ itself. As aforementioned, the function shall return a temporally conservative bound on the time between the maximum external enabling time of an iteration ι^{n-q+1}_i and the finish time of an iteration ι^n_i. For that reason it must consist of both an upper bound on the execution times of task τ_i itself, as well as an upper bound on the execution times of all higher priority tasks that can preempt and delay any of the executions of τ_i within the interval $[\hat{\varepsilon}(\iota^{n+q-1}_i), \hat{\varepsilon}(\iota^{n+q-1}_i) + w'_i(q)]$. Using the WCETs of the task τ_i, of the tasks $\tau_j \in hp(i)$ and the function $\gamma_{\tau_j \to w'_i(q)}$ we can consequently write $w'_i(q)$ as follows:

$$w'_i(q) = q \cdot C_i + \sum_{\tau_j \in hp(i)} \gamma_{\tau_j \to w'_i(q)} \cdot C_j$$

Note that $w'_i(q)$ is computed iteratively until a fixed-point is found. This is due to the fact that the larger $w'_i(q)$ becomes, the more higher priority tasks $\tau_j \in hp(i)$ can interfere with $w'_i(q)$, leading to an even larger $w'_i(q)$.

Next we can substitute $w_i(\varepsilon(\iota^{n-q+1}_i), q)$ in Equation 5 using Equation 6 and $w^*_i(\varepsilon(\iota^{n-q+1}_i))$ using Equation 7, which results in the following upper bound on the finish time of an iteration ι^n_i (with ε a shorthand notation for $\varepsilon(\iota^{n-q+1}_i)$):

$$\begin{aligned} \hat{f}(\iota^n_i) &= \max_{q \geq 1}(\max_{\varepsilon \in \mathcal{E}(\iota^{n-q+1}_i)} (\varepsilon + w_i(\varepsilon, q))) \\ &= \max_{q \geq 1}(\max_{\varepsilon \in \mathcal{E}(\iota^{n-q+1}_i)} (\varepsilon + \hat{s}_i + (n-q+1) \cdot P_i - \varepsilon + w'_i(q))) \\ &= \hat{s}_i + \max_{q \geq 1}(w'_i(q) - (q-1) \cdot P_i) + n \cdot P_i = \hat{f}_i + n \cdot P_i \end{aligned}$$

Note that due to the same reasoning as in [21] $w'_i(q)$ only has to be considered if it holds that $w'_i(q-1) > (q-1) \cdot P_i$. Furthermore, the dependence on specific external enabling times $\varepsilon(\iota^{n-q+1}_i)$ is removed, making the maximum finish time calculation practically usable. As $w'_i(q)$ is independent of a specific iteration ι^n_i it follows that $\hat{f}(\iota^n_i)$ is indeed a periodic upper bound on the finish time of a task τ_i with respect to both interference of higher priority tasks and self-interference. We can use this bound to compute the maximum firing durations required in step 3 of our analysis flow as $\hat{f}_i - \hat{s}_i$.

Finally, to guarantee that our flow terminates we require that maximum external enabling times increase monotonically throughout iterations of the analysis flow. Considering the LP in Equation 2 it can be seen that this holds if the maximum firing durations also increase monotonically. The term $-\hat{s}_i$ in the computation of $\gamma^*_{\tau_j \to w'_i(q)}$, however, can lead to decreasing maximum firing durations. For that reason we have to clamp the maximum firing durations. Let $\hat{\rho}^{k-1}_i$ be the maximum firing duration computed in iteration $k-1$ of the analysis flow. With $\hat{\rho}^0_i = C_i$ we can compute the maximum firing duration in iteration k as follows:

$$\begin{aligned} \hat{\rho}^k_i &= \max(\hat{\rho}^{k-1}_i, \hat{f}_i - \hat{s}_i) \\ &= \max(\hat{\rho}^{k-1}_i, \max_{q \geq 1}(w'_i(q) - (q-1) \cdot P_i)) \end{aligned}$$

E. Iterative Buffer Sizing

The iterative buffer sizing in our analysis flow makes use of the equations from [25], which are presented in this section. We denote a number of initially empty containers estimated in iteration k of the analysis flow as δ^k_{ji}. Numbers of empty containers are initialized to $\delta^0_{ji} = 1$ if $\delta_{ij} = 0$ and to $\delta^0_{ji} = 0$ otherwise. The reason for this assignment is that any smaller initial values would create cycles with zero tokens in the corresponding dataflow model, which would cause deadlock. Based on this the numbers of initially empty containers are estimated in step 3 of the analysis flow as follows:

$$\delta^k_{ji} = \begin{cases} \max(\left\lceil \frac{\hat{s}_j + \hat{\rho}_j - \check{s}_i}{P_j} \right\rceil, 0) & \text{buffer with non-blocking writes} \\ \max(\left\lceil \frac{\hat{s}_j + \hat{\rho}_j - \hat{s}_i}{P_j} \right\rceil, \delta^{k-1}_{ji}) & \text{buffer with blocking writes} \end{cases}$$

It is thereby differed between buffers with blocking writes, i.e. buffers that block a writing task whenever the buffer is full, and buffers with non-blocking writes. For buffers with blocking writes a clamping of initially empty containers by the initially empty containers of the previous iteration of the analysis flow is required. Otherwise, monotonicity and therefore termination of the flow would not be guaranteed.

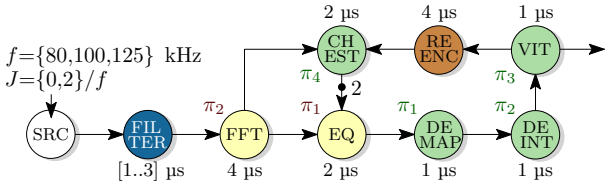

Fig. 6. HSDF graph of the packet decoder of a WLAN 802.11p transceiver.

V. Case Study

This section demonstrates the benefits of our approach in a case study. At first we discuss the performance of our approach for acyclic applications in more detail, again using the example in Figure 1 that was already briefly introduced in Section II. Then we address a more complex, cyclic application and relate our approach to other approaches applicable for cyclic applications. Finally, we discuss the run-times of our analysis approach in different scenarios.

A. Acyclic Applications

Let us reconsider the example depicted in Figure 1. It is no surprise that our approach and MAST compute the same end-to-end latencies for the presented task graph, as both rely on the same principles: Both approaches make use of a notion of dynamic offsets that is combined with an explicit exclusion of interference due to precedence constraints. In fact, sometimes MAST can even have slightly more accurate results than our approach for acyclic applications because it determines the critical instant and makes use of external enabling intervals. We make use of execution intervals and consequently do not need to compute critical instants. This comes at the cost of a slight over-approximation, but simplifies the combination with arbitrary (cyclic) precedence constraints and thus allows to perform a more accurate analysis for cyclic applications.

The SymTA/S approach has a significantly worse analysis accuracy than our approach for the given example. This is due to the fact that SymTA/S does not have a notion of explicit interference exclusion, which is especially relevant if large jitters, i.e. jitters that are larger than the source period, are involved. In some cases, however, e.g. if the priorities of tasks τ_i and τ_k were reversed [15], the SymTA/S approach can still outperform both our approach and MAST. The reason for this is that we make use of periodic bounds, which implies the assumption that bursts of input values that occur due to large input jitters are processed periodically, with the frequency of the source. Therefore our computed end-to-end latencies cannot be smaller than the input jitter plus the time to process one value. SymTA/S instead can take into account that a burst of multiple input values can be sometimes processed faster than with the source frequency, leading to smaller end-to-end latencies.

However, as it was already explained in Section II, we can apply an iterative buffer sizing on acyclic task graphs. This makes acyclic task graphs cyclic, such that approaches like MAST or SymTA/S are not applicable anymore. The iterative buffer sizing introduces additional backward dependencies, which are exploited by our algorithm to further limit interference between tasks. This can lead to more accurate analysis results than both MAST and SymTA/S can achieve.

B. Cyclic Applications

This section demonstrates the benefits of our approach for cyclic real-time streaming applications. We analyze the task graph of a WLAN 802.11p transceiver [1]. WLAN 802.11p transceivers are used in safety-critical automotive applications like automated braking systems, which imposes the requirement to give guarantees on the temporal behavior of such transceivers. A WLAN 802.11p transceiver has several modes and is executed on a multiprocessor system for performance reasons. According to the standard usually two identical transceivers are required, one for a control and one for a data channel, that must be executed in parallel. We only consider the part of the task graph that is active during packet decoding mode. An HSDF model corresponding to the task graph of the packet decoding mode used in both control and data parts is shown in Figure 6.

A periodic source models the input of this dataflow graph. The source frequency f at which the transceiver operates is typically 125 kHz. For illustration purposes, however, we vary the source frequency between 80 kHz, 100 kHz and 125 kHz. Furthermore, we consider an optional Direct Memory Access (DMA) unit between the A/D converter at the source and the filter task that can introduce bursts of up to three simultaneously arriving symbols. This behavior can be modeled by an input jitter of $J = 2/f$.

The dataflow graph contains a feedback loop as the settings of the channel equalizer (EQ) for the reception of symbol n are based on an estimate of the channel (CHEST). The channel estimate is in turn based on the received symbol $n-2$ and the reencoded symbol $n-2$, which is obtained by reencoding the error-corrected bits of symbol $n-2$ produced by the viterbi channel decoder (VIT).

The BCETs and WCETs of the tasks, which are denoted next to the corresponding dataflow actors, represent theoretical bounds on actual execution times on the Starburst platform [4]. The bounds adhere to the requirements given in Section IV-A, i.e. they are temporally conservative, independent of schedules and include communication times. On the Starburst platform these requirement can be satisfied if the considered processors are uncached, if SDRAM is not used and if communication scheduling is done in a round-robin fashion.

Our analysis flow allows for the quick verification of different task-to-processor mappings and priority assignments. In the following we assume that the tasks of both control and data parts of the transceiver are mapped to eight different processors. At first, we consider the case that the control and the data transceiver are executed independent from each other, i.e. the tasks of the control part are mapped to a different set of four processors than the tasks of the data part. One such mapping on four processors is exemplified in Figure 6, with different colors of actors indicating different processors. If multiple tasks are mapped to a shared processor (have the same color) then they are scheduled by a static priority preemptive scheduler, with the priorities denoted as π_1 (lowest) to π_4 (highest).

Due to the cyclic data dependency introduced by the feedback loop none of the offset-based approaches discussed in Section II is applicable on this example. Moreover, the approach in [9] is applicable in principal, but determines a constraint violation for all considered cases.

Consequently, we can only relate our approach to the one from [26], which combines a period-and-jitter interference characterization with an explicit consideration of precedence constraints (case PJ), and to the approach from [25], which additionally applies an iterative buffer sizing under the assumption that all used FIFO buffers block on writes (case PJ+iBS). For comparison we also apply our approach, which combines execution intervals with precedence constraints, with or without using the iterative buffer sizing (cases EI and EI+iBS). Finally, we verify the temporal conservativeness of

J / µs	0			25	20	16
f / kHz	80	100	125	80	100	125
PJ	$L = 25$ $\Delta = 13$	constraint violation	constraint violation	constraint violation	constraint violation	constraint violation
PJ+iBS	$L = 19$ $\Delta = 12$	$L = 19$ $\Delta = 13$	constraint violation	$L = 44$ $\Delta = 14$	$L = 39$ $\Delta = 15$	constraint violation
EI	$L = 12$ $\Delta = 12$	$L = 12$ $\Delta = 12$	$L = 14$ $\Delta = 13$	$L = 49$ $\Delta = 17$	constraint violation	constraint violation
EI+iBS	$L = 12$ $\Delta = 12$	$L = 12$ $\Delta = 12$	$L = 14$ $\Delta = 13$	$L = 37$ $\Delta = 14$	$L = 32$ $\Delta = 14$	$L = 30$ $\Delta = 15$
SIM	$L = 12$ $\Delta = 12$	$L = 12$ $\Delta = 12$	$L = 14$ $\Delta = 13$	$L = 29$ $\Delta = 14$	$L = 29$ $\Delta = 14$	$L = 28$ $\Delta = 15$

TABLE I. TEMPORAL ANALYSIS RESULTS FOR FIGURE 6 (L IN µs).

our results by comparing them to the times obtained from the high-level system simulator Hapi (case SIM). The Hapi simulator was initially a dataflow simulator [2], but was recently extended with the addition of processor sharing, which allows for the simulation of task graph executions on arbitrary platforms. Note that in our simulations we assume the same buffer capacities as computed in the case EI+iBS.

The different analysis approaches are applied for each combination of source jitter and frequency. The analysis results are then used to compute end-to-end latencies L and sums of all buffer capacities Δ. The respective end-to-end latencies and sums of buffer capacities of the control part are presented in Table I (the results for the data part of the transceiver are equal since both parts are executed on independent sets of processors). Note that for some cases the analyzed latencies grow so large that the throughput constraint imposed by the feedback loop is violated, which is indicated by the "constraint violation" entries in the table.

What immediately catches one's eye is that the combination of execution intervals, a consideration of precedence constraints and an iterative buffer sizing (case EI+iBS) produces the most accurate results, whereas the approach from [26] (case PJ) generates the worst. The usage of execution intervals allows for a more accurate interference characterization than achievable with period-and-jitter. This is due to the fact that in the period-and-jitter characterization all interfering tasks are assumed to be externally enabled simultaneously, whereas our approach rules out interference between tasks if no overlap between execution intervals and maximum busy periods is detected. The effect of using execution intervals on analysis accuracy becomes apparent by comparing the cases PJ and EI: Not only does our algorithm converge in more cases without a violation of temporal constraints, but also the computed end-to-end latencies are much smaller (up to 52% for $f = 80$ kHz and $J = 0$ µs). The simulation results (case SIM) are equal to our analysis results (case EI+iBS) if the input jitter is zero. This does not only confirm that our results are temporally conservative, but also that our analysis results are indeed very accurate in this case.

It can be seen that the iterative buffer sizing is especially helpful when the source jitter J becomes larger, as it effectively reduces interference between the tasks FFT and EQ by limiting the buffer capacity between the two. Moreover, our approach with iterative buffer sizing (case EI+iBS) produces consistently more accurate results than the approach from [25] (case PJ+iBS) (the computed end-to-end latency is up to 37% smaller). This shows that execution intervals can provide a more accurate interference characterization than achievable by an iterative buffer sizing only. The latter is especially true for the tasks DEMAP, DEINT and VIT on the feedback loop, whose finish times are severely overestimated in the cases

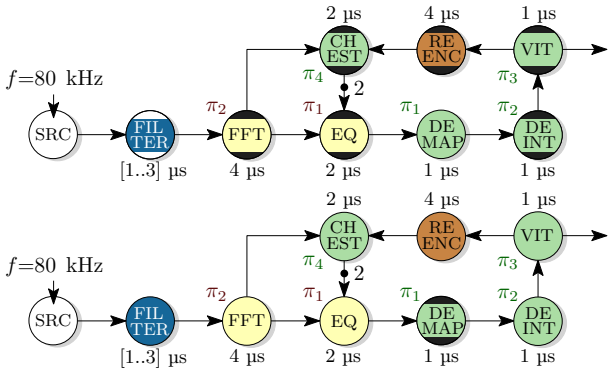

Fig. 7. Processor sharing between control and data parts of WLAN 802.11p transceiver packet decoders.

f / kHz	PJ	PJ+iBS	EI	EI+iBS	SIM
80	constraint violation	$L = 24$ $\Delta = 13$	$L = 20$ $\Delta = 12$	$L = 20$ $\Delta = 12$	$L = 16$ $\Delta = 12$

TABLE II. TEMPORAL ANALYSIS RESULTS FOR FIGURE 7 (L IN µs).

without execution intervals. Finally, simulation results (SIM) again confirm that our results are temporally conservative. The deviation between analysis and simulation results can be explained by the fact that our analysis uses periodic bounds and hence assumes that bursts of input values are processed periodically, with the period of the source, whereas actually a faster processing may be possible.

Now consider the example depicted in Figure 7, which represents the dataflow models of both control and data parts of the transceiver decoding modes. Most of the tasks are still executed in isolation, except for the DEMAP tasks, whose mappings are swapped between the control and data parts.

The analysis results for this example are presented in Table II, again for the control part only (results for the data part are also equal again, since the mappings of the DEMAP tasks are swapped symmetrically). At first, one can see that although the modification compared to Figure 6 is minimal, the results are much worse compared to the case that the task graphs are executed in isolation: Latencies and sums of buffer capacities are consistently larger than the ones presented in the first column of Table I. Moreover, the frequencies $f = 100$ kHz and $f = 125$ kHz, as well as jitters of $J = 2/f$, do not even appear in the table, as all considered approaches report a violation of constraints for these cases. Note that this difference does not only come from an analysis inaccuracy, but the fact that the DEMAP tasks become indeed uncorrelated with the other tasks on their processors. This means that the DEMAP tasks actually can experience more interference than in Figure 6, which becomes apparent by comparing the simulation results between Table II and the first column of Table I.

Finally, the results for execution intervals are still better than the results for period-and-jitter, although the gap between the results is smaller than in the single task graph case. The latter is due to the fact that for uncorrelated tasks the period-and-jitter characterization is very similar to the characterization described by $\gamma^*_{\tau_j \to w'_i(q)}$. In fact, in some cases period-and-jitter can be even slightly more accurate for interference over multiple task graphs. This, as well as the conclusion from the following section that run-times of interference computations are small compared to the overall algorithmic run-times, suggests that a combination of period-and-jitter and execution intervals can lead to an even better analysis accuracy if an application consists of multiple task graphs.

C. Analysis Run-Times

The presented analysis algorithms were executed on a PC with an Intel® Core™ i7 processor. All measured run-times are in the microsecond range, which shows that our approach is indeed capable of a quick temporal analysis for cyclic real-time streaming applications. Although the measured run-times are neglectably small, we can still make some observations about their influencing factors. The invocations of the algorithms PJ and EI for the graph in Figure 6 all take around 65 µs, independent of source frequencies and jitters. This implies that the interference computations only take marginal portions of the run-times, since no differences can be observed between jitter-and-period and the computationally more complex execution interval characterization. The combinations with an iterative buffer sizing take around 25% longer (around 80 µs in all cases), which can be explained by the additional, repetitive invocations of the Floyd-Warshall algorithm.

For the example in Figure 7 the run-times are in all cases less than 100% larger compared to the case in Figure 6 (around 110 µs without and around 160 µs with iterative buffer sizing), although the number of analyzed tasks is doubled. This indicates that the run-times scale well in numbers of tasks, allowing for the quick analysis of even larger sets of tasks than considered in this case study.

VI. Conclusion

This paper presented a temporal analysis approach for cyclic real-time stream processing applications executed on multiprocessor systems with processor sharing and static priority preemptive schedulers.

The approach introduces a notion of execution intervals, which are a generalization of dynamic offsets, to the temporal analysis of cyclic applications. It was shown that such execution intervals can be extracted from dataflow graphs that are used to model the best-case and worst-case temporal behavior of an analyzed application. Execution intervals are temporally conservative, which makes them suitable to bound interference between tasks on shared processors in a temporally conservative manner.

Furthermore, execution intervals were integrated with an explicit consideration of precedence constraints. While execution intervals are defined relative to executions of the source, precedence constraints establish a direct relation between interfering tasks. Taken together, execution intervals and precedence constraints produce an interference characterization that leads to a significant improvement of analysis accuracy for cyclic streaming applications, compared to state-of-the-art. Finally, if FIFO buffers with blocking writes are used for inter-task communication and if an iterative buffer sizing is applied on these buffers, then also significantly better analysis results can be achieved for acyclic applications.

The applicability and performance of the presented approach were evaluated in a case study using a WLAN 802.11p transceiver application. It was shown that if an iterative buffer sizing is applied up to 25% higher throughput guarantees and up to 37% smaller end-to-end latencies can be determined compared to state-of-the-art. If iterative buffer sizing is not fully applicable (for instance because the usage of FIFO buffers with blocking writes is not available) then even an improvement of up to 56% in throughput guarantees and up to 52% in end-to-end latencies can be observed.

References

[1] P. Alexander, D. Haley, and A. Grant. Outdoor mobile broadband access with 802.11. *IEEE Communications Magazine*, 45(11):108–114, 2007.

[2] M. Bekooij, S. Parmar, and J. van Meerbergen. Performance guarantees by simulation of process networks. In *SCOPES*, pages 10–19, 2005.

[3] G. Buttazzo. *Hard Real-Time Computing Systems*. Springer US, 2011.

[4] B. Dekens, P. Wilmanns, M. Bekooij, and G. Smit. Low-cost guaranteed-throughput communication ring for real-time streaming MP-SoCs. In *DASIP*, 2013.

[5] J. Falk et al. A generalized static data flow clustering algorithm for MPSoC scheduling of multimedia applications. In *EMSOFT*, pages 189–198, 2008.

[6] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano. MAST: modeling and analysis suite for real time applications. In *ECRTS*, pages 125–134, 2001.

[7] J. Hausmans. *Abstractions for aperiodic multiprocessor scheduling of real-time stream processing applications*. PhD thesis, University of Twente, 2015.

[8] J. Hausmans, M. Bekooij, and H. Corporaal. Resynchronization of cyclo-static dataflow graphs. In *DATE*, pages 1–6, 2011.

[9] J. Hausmans, M. Wiggers, S. Geuns, and M. Bekooij. Dataflow analysis for multiprocessor systems with non-starvation-free schedulers. In *SCOPES*, pages 13–22, 2013.

[10] R. Henia and R. Ernst. Improved offset-analysis using multiple timing-references. In *DATE*, pages 450–455, 2006.

[11] R. Henia et al. System level performance analysis – the SymTA/S approach. *IEE Proc. of Computers and Digital Techniques*, 152(2):148–166, 2005.

[12] J. Kim et al. A novel analytical method for worst case response time estimation of distributed embedded systems. In *DAC*, pages 129:1–129:10, 2013.

[13] J. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, pages 26–37, 1998.

[14] J. Palencia and M. Gonzalez Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *RTSS*, pages 328–339, 1999.

[15] S. Perathoner et al. Influence of different abstractions on the performance analysis of distributed hard real-time systems. *Design Automation for Embedded Systems*, 13(1-2):27–49, 2008.

[16] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *ECRTS*, pages 239–248, 2004.

[17] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. CRC Press, 2009.

[18] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *DAC*, pages 777–782, 2007.

[19] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *DAC*, pages 899–904, 2006.

[20] K. Tindell. *Adding time-offsets to schedulability analysis*. University of York, Department of Computer Science, 1994.

[21] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[22] M. Wiggers, M. Bekooij, M. Geilen, and T. Basten. Simultaneous budget and buffer size computation for throughput-constrained task graphs. In *DATE*, pages 1669–1672, 2010.

[23] M. Wiggers, M. Bekooij, and G. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *SCOPES*, pages 11–22, 2007.

[24] M. Wiggers, M. Bekooij, and G. Smit. Computation of buffer capacities for throughput constrained and data dependent inter-task communication. In *DATE*, pages 640–645, 2008.

[25] P. Wilmanns, S. Geuns, J. Hausmans, and M. Bekooij. Buffer sizing to reduce interference and increase throughput of real-time stream processing applications. In *ISORC*, pages 623–630, 2015.

[26] P. Wilmanns, J. Hausmans, S. Geuns, and M. Bekooij. Accuracy improvement of dataflow analysis for cyclic stream processing applications scheduled by static priority preemptive schedulers. In *DSD*, pages 9–18, 2014.

[27] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Trans. on Parallel and Distributed Systems*, 9(11):1125–1136, 1998.