Temporal Analysis of Static Priority Preemptive Scheduled Cyclic Streaming Applications using CSDF Models

Philip S. Kurtin*
philip.kurtin@utwente.nl

*University of Twente, Enschede, The Netherlands

Marco J.G. Bekooij*‡
marco.bekooij@nxp.com

‡NXP Semiconductors, Eindhoven, The Netherlands

ABSTRACT

Real-time streaming applications with cyclic data dependencies that are executed on multiprocessor systems with processor sharing usually require a temporal analysis to give guarantees on their temporal behavior at design time. Current accurate analysis techniques for cyclic applications that are scheduled with Static Priority Preemptive (SPP) schedulers are however limited to the analysis of applications that can be expressed with Homogeneous Synchronous Dataflow (HSDF) models, i.e. in which all tasks operate at a single rate. Moreover, it is required that both input and output buffers synchronize atomically at the beginnings and finishes of task executions, which is difficult to realize on many existing hardware platforms.

This paper presents a temporal analysis approach for cyclic real-time streaming applications executed on multiprocessor systems with processor sharing and SPP scheduling that can be expressed using Cyclo-Static Dataflow (CSDF) models. This allows to model tasks with multiple phases and changing rates and furthermore resolves the problematic restriction that buffer synchronization must occur atomically at the boundaries of task executions. For that purpose a joint interference characterization over multiple phases is introduced, which realizes a significant accuracy improvement compared to an isolated consideration of interference.

Applicability, efficiency and accuracy of the presented approach are evaluated in a case study using a WLAN 802.11p transceiver application. Thereby different use-cases of CSDF modeling are discussed, including a CSDF model relaxing the requirement of atomic synchronization.

1. INTRODUCTION

Real-time stream processing applications that are executed on multiprocessor systems require guarantees on their temporal behavior. These guarantees must be already given at design time, in order to ensure that throughput and latency constraints can be always satisfied. A temporal analysis that can provide such guarantees is usually not trivial as the temporal behavior of an analyzed application is influenced by both cyclic data dependencies and processor sharing with run-time scheduling.

Cyclic data dependencies occur in applications with feedback loops. Moreover, inter-task communication is often realized via First-In-First-Out (FIFO) buffers with blocking writes. On a blocking write buffer it does not only hold that a reading task must wait if the buffer is empty, but also a writing task gets suspended if the buffer is full, resulting in additional cyclic data dependencies.

DOI: http://dx.doi.org/10.1145/2993452.2993564

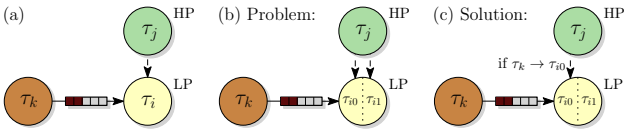

Figure 1: Basic idea of accurately considering interference for CSDF-type applications.

It has been shown that dataflow analysis techniques can be used for temporal analysis under such challenging circumstances. Especially the inherent support of cyclic data dependencies distinguishes dataflow analysis from other approaches. Besides temporal analysis, dataflow analysis techniques can be used for the computation of required buffer capacities [19], for the determination of scheduler settings [20], for finding suitable task-to-processor assignments [18] and for establishing a basis for synchronization overhead minimization techniques such as task clustering [4] and resynchronization [7].

One of the most challenging types of run-time schedulers with respect to temporal analyzability is the Static Priority Preemptive (SPP) scheduler [3] for which static priorities are assigned to tasks sharing a processor and for which higher priority tasks can preempt and thus delay lower priority tasks whenever they are enabled. In [8] an iterative algorithm is proposed that combines dataflow modeling with classical real-time analysis techniques, enabling the analysis of applications with cyclic data dependencies and SPP scheduling. This approach is extended in [21] by making use of the fact that cyclic data dependencies limit interference, resulting in a significantly higher analysis accuracy.

However, both [8] and [21] are limited to the analysis of applications that can be expressed with Homogeneous Synchronous Dataflow (HSDF) graphs, i.e. applications in which all tasks operate at the same rate. Moreover [21] requires that task synchronization happens atomically on the boundaries of tasks, which is difficult, if not impossible to realize on many existing hardware platforms. As we discuss later in this paper both these restrictions can be removed by supporting the analysis of applications that can be expressed using Cyclo-Static Dataflow (CSDF) models, i.e. applications in which tasks can be divided into multiple phases operating at different constant rates.

Note that methods exist to translate CSDF graphs into HSDF graphs with the same temporal behavior. This makes [8] and [21] applicable for CSDF-type applications. But the part of these algorithms which computes so-called maximum response times to include the effects of processor sharing would consider interference of higher priority tasks on each task phase in separation, i.e. treat each phase as a separate task, which would inevitably lead to overly pessimistic results. This is illustrated in Figure 1. Figure 1(a) depicts a higher priority (HP) task τ_j that can interfere once with a lower priority (LP) task τ_i, as well as a task τ_k that enables

task τ_i. If task τ_i is split into two phases, as depicted in Figure 1(b), of which now only phase τ_{i0} is enabled by τ_k, the existing algorithms would consider interference of τ_j for each phase separately and thus twice, once for the maximum response time of τ_{i0} and once for the one of τ_{i1}. From this follows that while existing analysis methods are applicable for CSDF in principle, the obtained analysis results are bound to become highly pessimistic, if not entirely useless.

In this paper we propose a temporal analysis algorithm that combines dataflow modeling with real-time analysis techniques and that is suitable for an accurate analysis of cyclic real-time stream processing applications expressible with CSDF models and scheduled using SPP. The main contribution is the introduction of a novel response time analysis technique for sequences of task phase executions that prevents accounting for the same interference multiple times, as illustrated in Figure 1(c). The technique takes into account that for run-time scheduling a task phase can be either externally enabled by another task or be in consecutive execution with preceding task phases of the same task. In the first case (e.g. for τ_{i0} following τ_k) interference must be considered fully as the external enabling is independent of previous interference considerations. However in the second case (e.g. for τ_{i1} following τ_{i0} or τ_{i0} following τ_{i1}) interference already taken into account for predecessors can be ignored because it does not matter in which phase the interference is considered, the finish time would remain the same. Finally, the presented analysis technique is extended with a joint interference characterization over multiple phases which exploits that cyclic data dependencies limit interference between tasks, resulting in a significantly higher analysis accuracy.

The remainder of this paper is structured as follows. Section 2 defines the CSDF model and Section 3 discusses the relation between analyzed applications and the model. The abstractions applied in our approach are introduced in Section 4. Section 5 presents the analysis flow and Section 6 introduces our technique for computing maximum response times by considering interference due to SPP scheduling jointly over multiple phases. Section 7 describes the dataflow analysis used to derive periodic bounds on task schedules, as well as maximum enabling jitters, that are both needed for the maximum response time computation. Section 8 discusses applicability, efficiency and accuracy of our algorithm by means of a case study. Section 9 presents related work and Section 10 finally states the conclusions.

2. ANALYSIS MODEL

We make use of CSDF graphs to calculate lower bounds on the best-case and upper bounds on the worst-case schedule of an analyzed application, as well as to determine cyclic data dependencies between tasks. The bounds on schedules are used for the verification of temporal constraints and the derivation of upper bounds on the jitter of tasks, whereas the cyclic data dependencies are used to limit interference that occurs due to processor sharing.

A CSDF graph is a directed graph $G = (V, E)$ 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 consists of several distinct phases v_{ix} with $x \in \{0 \dots \theta_i - 1\}$, forming the cyclo-static period of an actor. Each of the phases is assigned a firing duration ρ_{ix}, a consumption rate γ_{jix} for each input edge e_{ji} and a production rate π_{ixj} for each output edge e_{ij}.

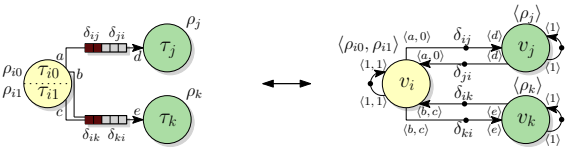

Figure 2: Task graphs and corresponding CSDF models.

The firing rule of a CSDF actor is as follows: A phase v_{ix} is enabled if all input edges e_{ji} contain at least γ_{jix} tokens. On an enabling the phase atomically consumes γ_{jix} tokens from all input edges e_{ji} and, after the firing duration of ρ_{ix}, atomically produces π_{ixj} tokens on all output edges e_{ij}.

Besides CSDF we also make use of HSDF models that are basically CSDF models in which all actors only have one phase and in which all rates are equal to one.

3. MODELING TASK GRAPHS WITH CSDF

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. Tasks read from and write to such buffers at predefined rates, which specify how many values are read or written per execution. We allow a task to consist of multiple phases that can have different rates. Our analysis requires Best-Case Execution Times (BCETs) and Worst-Case Execution Times (WCETs) of all task phases. Furthermore, we consider data-driven scheduling and thus require that a task phase is externally enabled, i.e. put in the ready queue of the scheduler, as soon as sufficient data is available in all its input buffers and sufficient space in all its output buffers, according to the predefined rates. Finally we assume that each task graph is triggered by a strictly periodic source producing one value per execution, which allows that the source can be modeled by a CSDF actor with a single phase and an output rate of one (note that a cyclo-static source with varying rates can be expressed with a virtual CSDF task right after the source).

Writing data to an output buffer can be implemented with the following three steps. At first it is verified whether sufficient consecutive output buffer locations are not locked by a reading task phase. If this is not the case, the locations are locked by the writing task phase (acquisition of space). This is followed by the actual write operation to the locked buffer locations (data write) and finalized by unlocking the buffer locations, making them available to reading task phases 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.

We use CSDF graphs to model such task graphs, as exemplified in Figure 2. The depicted tasks consist of multiple cyclo-static phases which have different execution times and access different buffers at different constant rates. Each phase of a task is mapped to a phase of a CSDF actor. The firing durations of the actor phases are thereby set to vary between the BCETs and so-called maximum response times (WCETs extended by interference due to processor sharing) of the corresponding task phases. Moreover, a self-edge with one token and all rates being equal to one is added to each CSDF actor to model that the phases of a task are executed one after the other. Note that in the following we leave such self-edges implicit if they contain a single token and if the rates of all phases are one.

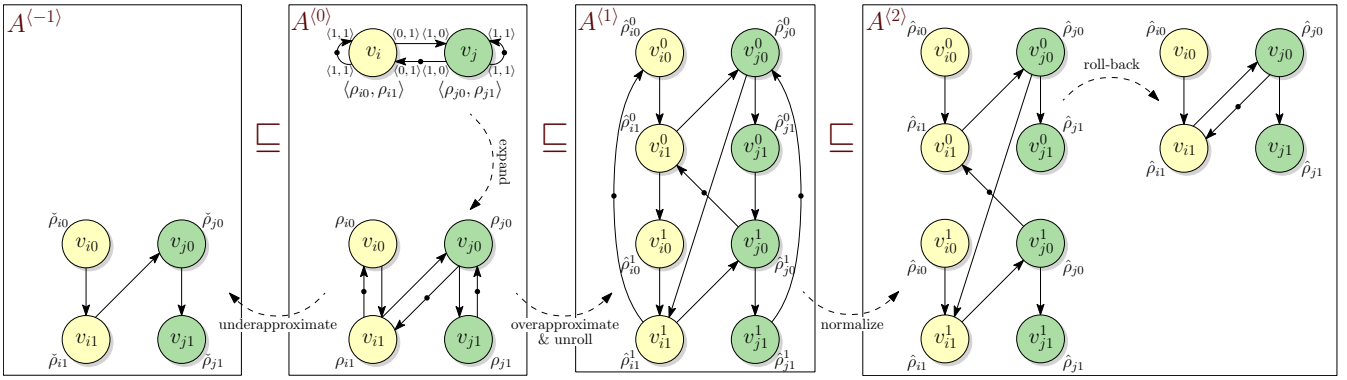

Figure 3: Abstraction levels used in our analysis flow.

A FIFO buffer is mapped to a pair of edges, one in the forward and one in the backward direction. The number of initial tokens on the forward edge is set to the number of initially full containers of the corresponding buffer and the number of initial tokens on the backward edge to the number of initially empty containers. The rates of a task phase are also mapped one-to-one to the rates of the corresponding actor phase on both forward and backward edges, with a rate of zero if a task phase does not access a buffer. The consumption of tokens by an actor phase v_{ix} from a forward edge then corresponds to an acquisition of data and from a backward edge to an acquisition of space. Analogously, a production of tokens corresponds to a release of data on a forward edge and a release of space on a backward edge.

In Section 8 some use-cases of modeling tasks by CSDF actors are shown. These cases include task clustering, the usage of a more fine-grained task model in which a task is separated into multiple synchronous sections and the introduction of a synchronization jitter. The latter two are of special importance as they both relax the scarcely satisfiable requirement of existing works such as [21] that all acquires of a task execution must happen atomically at its beginning and all releases atomically at its end.

4. ABSTRACTION LEVELS

Our analysis flow makes use of several so-called abstraction levels that comply with the the-earlier-the-better refinement theory presented in [6, 10] and that are illustrated for a part of a task graph in Figure 3. These levels are used to establish a strong relation between reality and our models such that the temporal behaviors that can occur in reality are conservatively overapproximated. Note that $A \sqsupseteq A'$ indicates that A is a temporally conservative abstraction of A' in the sense of the-earlier-the-better refinement.

The abstraction level $A^{\langle 0 \rangle}$ represents a so-called model of reality, as it is introduced in [15]. The model consists of one or more CSDF graphs that are derived according to Section 3. In [2] it is described how CSDF graphs can be expanded to HSDF graphs in which all actor phases of the CSDF model appear as separate actors and in which all actors have a rate of one. Such an expansion is exemplified in the lower half of abstraction level $A^{\langle 0 \rangle}$ and is required as our analysis techniques are in fact only directly applicable on HSDF graphs. The HSDF expansion on level $A^{\langle 0 \rangle}$ is used to derive the cyclic data dependencies which we use to limit interference as described in Section 6.4.

If rate conversions occur in the CSDF model, i.e. there exist edges with output rates not being equal to input rates, then r_i replications of all phases of a CSDF actor v_i occur in the HSDF expansion, resulting in $\Theta_i = r_i \cdot \theta_i$ HSDF actors per CSDF actor v_i. According to Section 3 the strictly periodic source of a task graph can be modeled by a CSDF actor with a single phase and a constant firing duration ρ_{s0} equal to its period. Assuming that in the HSDF expansion the source phase is replicated r_s times we can define the source period for the HSDF expansion as $P_s = r_s \cdot \rho_{s0}$. This allows us to define the period of all HSDF actors modeling a CSDF actor v_i triggered by that source as $P_i = P_s$. Note that for simplicity we do not differ between CSDF phases and replications in the following, but simply refer to a phase to indicate a single replication of a CSDF phase, as well as the corresponding task phase.

Abstraction level $A^{\langle 0 \rangle}$ is non-deterministic as the firing durations ρ_{ix} of actor phases can vary between BCETs and maximum response times of the corresponding tasks, with the latter being furthermore unknown in general. To perform our analysis we require a temporally conservative abstraction of this model in which all firing durations are constant and represent upper bounds on these maximum response times.

Simply replacing the varying, unknown firing durations ρ_{ix} on level $A^{\langle 0 \rangle}$ by constant upper bounds on maximum response times $\hat{\rho}_{ix} \geq \rho_{ix}$ does not suffice, though. The reason is that higher priority tasks can have bursts, resulting in the response times of lower priority task phases becoming temporarily very large. Using such response times as constant firing durations of the corresponding actor phases could lead to firing durations of entire actors becoming constantly larger than the source period. This would result in so-called self-delay due to the self-edges with one token and the dataflow analysis would consequently report a constraint violation.

Fortunately the effects of bursts average themselves out with time, such that response times of tasks eventually get lower than the source period again. This is exploited on abstraction level $A^{\langle 1 \rangle} \sqsupseteq A^{\langle 0 \rangle}$ in which the expanded graph on level $A^{\langle 0 \rangle}$ is unrolled until the total firing duration of all unrollings becomes smaller than the source period times the number of unrollings. The firing durations of the phases in the first unrolling $\hat{\rho}^0_{ix}$ are thereby derived under the assumption that during the corresponding task phase executions all higher priority task phases have maximum bursts, whereas the firing durations of the phases in subsequent unrollings model the averaging-out of these bursts and are consequently smaller, i.e. it holds with q^* the last unrolling that $\forall_{0<q\leq q^*}: \ \hat{\rho}^q_{ix} \leq \hat{\rho}^0_{ix}$. This results in a model that is deterministic due to the constant firing durations and in which no constraint is violated due to self-delay, as self-delay can only occur between unrollings q and $q+1$, with $0 \leq q < q^*$, but not between unrollings q^* and 0. The unrolling expressed in this model is used locally, i.e. on a per task basis, for the determination of maximum response times presented in

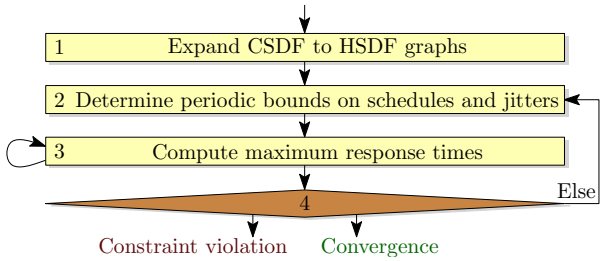

Figure 4: Overview of the analysis flow.

Section 6. However, using the same model for deriving delays between different tasks due to data dependencies as discussed in Section 7 would impose a significant complexity problem, as on each unrolling not only the phases of one actor must be unrolled, but all phases of all actors of the entire graph.

To address this complexity problem we introduce another abstraction level $A^{\langle 2 \rangle} \sqsupseteq A^{\langle 1 \rangle}$ on which we apply a process we call normalizing. In a first step we remove all self-edges connecting different unrollings of an actor. To maintain temporal conservativeness we include the maximum delays on the enablings of the first actor phases that can occur due to the removed edges in the firing durations of these actors phases, resulting in firing durations $\hat{\rho}_{i0}^{'k} \geq \hat{\rho}_{i0}^{k}$. For the other phases no self-edges are removed, such that we can assign $\forall_{0<x<\Theta_i}\colon\ \hat{\rho}_{ix}^{'k} = \hat{\rho}_{ix}^{k}$. In a second step we set all firing durations of actor phases on level $A^{\langle 2 \rangle}$ to the maximum of these enlarged firing durations belonging to different unrollings, i.e. $\hat{\rho}_{ix} = \max_{0 \leq k < n}(\hat{\rho}_{ix}^{'k})$.

This process leads to the depicted graph without self-edges between unrollings and the same firing durations for all unrolled versions of the same actor phases. Consequently we can roll this graph back to the expanded graph on the right side of level $A^{\langle 2 \rangle}$ that has the same complexity as the expanded graph on level $A^{\langle 0 \rangle}$. We use such rolled-back graphs in Section 7 to derive upper bounds on the schedules of task phases.

In Section 7 we compute bounds on the enabling jitters of task phases. Therefore we do not only require an abstraction of reality to compute upper bounds on the enabling times of task phases, but also a refinement of reality to compute lower bounds on enabling times. Such a refinement is presented on abstraction level $A^{\langle -1 \rangle} \sqsubseteq A^{\langle 0 \rangle}$ on which the constant firing durations of actor phases are set to the BCETs of the corresponding task phases, such that $\check{\rho}_{ix} \leq \rho_{ix}$. Moreover all edges containing tokens are removed, as edges with tokens can lead to delays in the model that do not have to occur in reality.

5. ANALYSIS FLOW

Figure 4 depicts the flow of our temporal analysis approach which we use to give guarantees on the temporal behavior of applications. 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.

In step 1 a CSDF model is determined which corresponds to the input task graphs as discussed in Section 3, thus complying with abstraction level $A^{\langle 0 \rangle}$ in Section 4. This CSDF model is then expanded to a corresponding HSDF model, using the method described in [2].

Step 2 computes two periodic schedules, one a lower bound on the best-case behavior of the task phases and the other an upper bound on their worst-case behavior. The best-case schedule is computed using the expanded model on abstraction level $A^{\langle -1 \rangle}$ and the worst-case schedule using the rolled-back, but expanded model on level $A^{\langle 2 \rangle}$. The firing durations in the best-case model are set to BCETs, while the firing durations in the worst-case model are initialized to WCETs and in later iterations set to the maximum response times computed in step 3. With these periodic schedules upper bounds on the enabling jitters of task phases are determined.

In step 3 maximum response times of task phases are computed that are normalized using the method described in Section 4 to derive firing durations for the actor phases on abstraction level $A^{\langle 2 \rangle}$. To include the effects of processor sharing in the maximum response times two interference characterizations are considered, one based on maximum enabling jitters and periods and the other on cyclic data dependencies derived from the expanded version of abstraction level $A^{\langle 0 \rangle}$.

Note that the maximum response times computed in step 3 depend on the schedules and jitters computed in step 2, which themselves depend on the maximum response times computed in step 3. This mutual dependency is the reason for the iterative character of our analysis flow. Consequently, we check in step 4 whether all periodic schedules have converged, i.e. did not change since the last iteration of the analysis flow, or whether any temporal constraint is violated. If none of this is the case, the iterative loop is repeated, starting again with step 2, until either convergence or constraint violation is achieved. All steps of the analysis flow are monotone, i.e. their results cannot decrease throughout increasing iterations of the flow, which is a necessary requirement for convergence.

6. MAXIMUM RESPONSE TIMES

In this section we introduce our method to compute maximum response times of individual task phases in step 3 of the analysis flow. At first we present a state-of-the-art algorithm for the computation of maximum response times of tasks in Section 6.1. Its shortcomings and the basic idea of our approach are discussed in Section 6.2. Section 6.3 extends the state-of-the-art algorithm to an algorithm for the computation of maximum response times of task phases. Finally Section 6.4 presents the derivation of an interference characterization for multiple task phases which exploits the effect that cyclic data dependencies limit interference.

Note that in the remainder of this paper we use the terms upper bound on the worst-case (lower bound on the best-case) and maximum (minimum) interchangeably. Moreover we use the shorthand notation ι_{ix}^{n} to denote both the n'th firing of an actor phase v_{ix} and the n'th execution of a task phase τ_{ix}.

6.1 State-of-the-Art

Before we introduce our algorithm to compute maximum response times of task phases let us first recap the equations from [21] that are used to derive maximum response times $\hat{\rho}_i$ of entire tasks (or, in other words, tasks that consist of only one phase). In the following we differ between the external enabling and the internal enabling time of a task execution. A task execution is externally enabled once there is sufficient data in its input buffers and sufficient space in its output buffers, whereas it is internally enabled once its previous execution is finished. A necessary requirement to derive a maximum response time of a task τ_i is the existence of a periodic upper bound on the external enabling times $\varepsilon^{ext}(\iota_i^n)$ of all executions of that task, i.e.:

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

Given such a bound it is shown that a periodic upper bound on the finish time of a task execution ι_i^n can be determined by computing maximum response times $\hat{\rho}_i$ as follows:

$$\forall_{n\geq 0}\colon\ f(\iota_i^n) \leq \hat{f}(\iota_i^n) = \hat{\varepsilon}^{ext}(\iota_i^n) + \underbrace{\max_{q\geq 0}(w_i(q) - q\cdot P_i)}_{=\hat{\rho}_i} \quad (1)$$

with

$$w_i'(q) = (q+1)\cdot C_i + \sum_{j\in hp(i)} \eta_j(w_i'(q))\cdot C_j$$

$$w_i(q) = (q+1)\cdot C_i + \sum_{j\in hp(i)} \gamma_{j\to i}(w_i'(q), q)\cdot C_j$$

and

$$\eta_j(\Delta t) = \left\lceil \frac{\hat{J}_j + \Delta t}{P_j} \right\rceil,\ \zeta_{j\to i}(q) = \delta(\mathcal{P}_{ij}) + \delta(\mathcal{P}_{ji}) + q - 1$$

$$\gamma_{j\to i}(\Delta t, q) = \min(\eta_j(\Delta t), \zeta_{j\to i}(q))$$

The computation makes use of so-called maximum busy periods $w_i'(q)$ and $w_i(q)$ that are defined as upper bounds on the total execution time of $q+1$ executions of a task τ_i. This implies that not only the WCET C_i of task τ_i is included $q+1$ times, but also the sum of all possible interferences of higher priority tasks τ_j with $j \in hp(i)$. The interference of a task τ_j is thereby computed as the maximum number of executions of τ_j that can occur within $w_i(q)$, multiplied by the corresponding WCET of task τ_j. Note that we have opted for indicating the first considered execution with $q = 0$, whereas existing works begin with $q = 1$.

In a first step a maximum busy period $w_i'(q)$ is determined based on the $\eta_j(\Delta t)$ only. This interference characterization gives the maximum number of executions of a task τ_j in any time interval Δt, using the maximum enabling jitter $\hat{J}_j$ and the period P_j. The maximum busy period $w_i'(q)$ is computed iteratively until a fixed point is found, which is required due to the dependency of $\eta_j(w_i'(q))$ on $w_i'(q)$.

Afterwards the maximum busy period $w_i'(q)$ is reduced to $w_i(q)$ by computing interference not only based on period-and-jitter, but also based on cyclic data dependencies. The function $\zeta_{j\to i}(q)$ gives the maximum number of executions of a task τ_j during q executions of a task τ_i based on cyclic data dependencies between the two in the corresponding HSDF model. For that means the minimum numbers of initial tokens on any path from (to) an actor v_i to (from) an actor v_j are determined, which are denoted as $\delta(\mathcal{P}_{ij})$ and $\delta(\mathcal{P}_{ji})$, respectively.

The reason for not only considering one execution of task τ_i (i.e. $q = 0$), but multiple executions, is that the maximum finish time $\hat{f}(\iota_i^n)$ is defined relative to the maximum external enabling time $\hat{\varepsilon}^{ext}(\iota_i^n)$. Consequently any delays of previous executions must be included in $\hat{\rho}_i$, which is achieved by assuming that an execution of task τ_i can be in consecutive execution with any number q executions of its predecessors. To maximize such self-delay it is further assumed that also the first of $q+1$ executions is enabled externally at its maximum external enabling time. Recall that the maximum external enabling time is periodic with P_i which explains the term $-q\cdot P_i$ in Equation 1. Finally it can be seen that a $w_i(q+1)$ only has to be considered if it holds that $w_i'(q) > (q+1)\cdot P_i$.

6.2 Basic Idea

Consider the HSDF graph depicted on the left-hand side of Figure 5. The WCETs of the corresponding tasks are denoted next to the actors. We apply the state-of-the-art algorithm presented in the previous section to compute an upper bound on the finish time of task τ_i. From the properties given in the caption of the figure it follows that we only have to consider $w_i(0)$ and that higher priority task τ_j can only interfere once with lower priority task τ_i, resulting in the maximum finish time denoted below the graph.

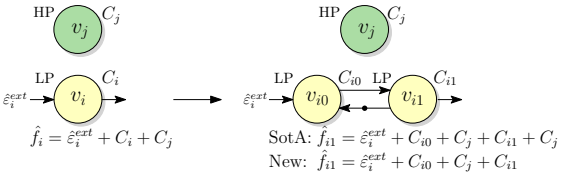

Figure 5: Basic idea (with $P_i = P_j \geq C_i + 2\cdot C_j$, $\hat{J}_j = 0$, $C_i = C_{i0} + C_{i1}$).

Let us now split the task τ_i into two phases, as depicted on the right-hand side of the figure. As the algorithm from the previous section does not have a notion of phases it treats each phase as a separate task. This results in a maximum finish time of $\hat{f}_{i0} = \hat{\varepsilon}_i^{ext} + C_{i0} + C_j$ for τ_{i0} which is at the same time the maximum external enabling time of task phase τ_{i1}. As the phases are treated as separate tasks also the maximum finish times are computed separately, such that $\hat{f}_{i1} = \hat{f}_{i0} + C_{i1} + C_j$, resulting in the upper maximum finish time in the figure. Due to the separate computation of maximum response times one can see that the interference of task τ_j is considered twice and thus overapproximated.

To resolve this problem we propose to compute maximum busy periods over multiple phases. For phase τ_{i0} we compute $w_{i0\to i0} = C_{i0} + C_i$. For phase τ_{i1} we then extend this maximum busy period to $w_{i0\to i1}$ by C_{i1} and any interference that can occur during this extension, i.e. interference that can occur during the whole maximum busy period over both τ_{i0} and τ_{i1} minus the interference already considered for τ_{i0}. This is allowed as the phases τ_{i0} and τ_{i1} are always in consecutive execution, which implies that if an interference of C_j occurred during the execution of τ_{i1} the internal enabling time of τ_{i1} would be at the same time reduced by C_j. As no additional interference can occur in the extension we derive the lower maximum finish time in the figure, correctly considering only one interference of task τ_j.

Note that if phase τ_{i1} were additionally externally enabled by another task τ_k we would have to consider a second maximum busy period $w_{i1\to i1}$ starting at external enabling time $\hat{\varepsilon}_{i1}^{ext}$. For $w_{i1\to i1}$ we would not be allowed to exclude interference considered in $w_{i0\to i0}$ as $\hat{\varepsilon}_{i1}^{ext}$ does not get smaller if the interference of τ_j is assumed to occur during τ_{i1} instead of τ_{i0}. This would result in $w_{i1\to i1} = C_{i1} + C_j$ and a finish time of phase τ_{i1} being equal to $\hat{f}_{i1} = \max(\hat{\varepsilon}_{i0}^{ext} + w_{i0\to i1}, \hat{\varepsilon}_{i1}^{ext} + w_{i1\to i1})$. Moreover, for smaller periods we would also need to consider self-delay, which can be achieved by extending both maximum busy periods $w_{i0\to i1}$ and $w_{i1\to i1}$ over additional executions of τ_{i0} and τ_{i1}.

The derivation of an algorithm that is capable of computing maximum response times using such maximum busy periods over multiple phases is subject to the next section.

6.3 Maximum Response Times of Task Phases

To compute accurate maximum response times of task phases we first translate the maximum finish time computation in Equation 1 into an algorithm. Subsequently we describe the necessary extensions of this algorithm for computing maximum busy periods over multiple task phases. Finally we describe the derivation of maximum response times of task phases from the maximum finish times computed by this algorithm.

The algorithm presented in Figure 6 produces the same results as Equation 1 if we redefine the interference characterizations for zero inputs, such that $\eta_i(0) = 0$ and $\gamma_{j\to i}(0, -1) = 0$. This is allowed as no interference has to be considered if there is nothing to interfere with.

1 $\hat{f}_i = 0$;
2 $q = 0$; $w_i^{'} = w_i = 0$;
3 do {
4 $w_i^{\oplus'} = C_i + \sum_{j \in hp(i)} [\eta_j(w_i^{'} + w_i^{\oplus'}) - \eta_j(w_i^{'})] \cdot C_j$;
5 $w_i^{\oplus} = C_i + \sum_{j \in hp(i)} [\gamma_{j \to i}(w_i^{'} + w_i^{\oplus'}, q) - \gamma_{j \to i}(w_i^{'}, q-1)] \cdot C_j$;
6 $w_i^{'} = w_i^{'} + w_i^{\oplus'}$; $w_i = w_i + w_i^{\oplus}$;
7 $\hat{f}_i = \max(\hat{f}_i, \hat{s}_i^{ext} + w_i - q \cdot P_i)$;
8 $q++$;
9 } while $(w_i^{'} > q \cdot P_i)$;

Figure 6: Algorithm to compute upper bounds on finish times of tasks.

1 $\forall_{0 \le x < \Theta_i}$: $\hat{f}_{ix} = 0$;
2 forall $(x : e_{jyix} \in E^{ext})$ {
3 $x' = x$; $q = 0$; $w_{ix}^{'} = w_{ix} = 0$; $\mathcal{Z}_{ix} = \emptyset$;
4 do {
5 $w_{ix}^{\oplus'} = C_{ix'} + \sum_{jy \in hp(i)} [\eta_{jy}(w_{ix}^{'} + w_{ix}^{\oplus'}) - \eta_{jy}(w_{ix}^{'})] \cdot C_{jy}$;
$w_{ix}^{\oplus} = C_{ix'} + \sum_{jy \in hp(i)} [\gamma_{jy}(w_{ix}^{'} + w_{ix}^{\oplus'}, \mathcal{Z}_{ix} \cup \{(v_{ix'}, q)\})$
6 $- \gamma_{jy}(w_{ix}^{'}, \mathcal{Z}_{ix})] \cdot C_{jy}$;
7 $w_{ix}^{'} = w_{ix}^{'} + w_{ix}^{\oplus'}$; $w_{ix} = w_{ix} + w_{ix}^{\oplus}$;
8 $\mathcal{Z}_{ix} = \mathcal{Z}_{ix} \cup \{(v_{ix'}, q)\}$;
9 $\hat{f}_{ix'} = \max(\hat{f}_{ix'}, \hat{s}_{ix}^{ext} + w_{ix} - q \cdot P_i)$;
10 $x'++$;
11 if $(x' = \Theta_i)$ {
12 $q++$; $x' = 0$;
13 }
14 } while $(x' \neq x \;||\; w_{ix}^{'} > q \cdot P_i)$;
15 }

Figure 7: Algorithm to compute upper bounds on finish times of task phases.

Given upper bounds on the external enabling times of task phases $\hat{s}_{ix}^{ext}$, which are derived in step 2 of the analysis flow, we can now extend the finish time bound for tasks in Figure 6 to the finish time bound for task phases presented in Figure 7. The interference characterization $\eta_{jy}(\Delta t)$ used in this algorithm is the same as $\eta_j(\Delta t)$ in Equation 1, except for the difference that $\hat{J}_j$ is replaced by a maximum enabling jitter per task phase $\hat{J}_{jy}$. This is allowed as it does not matter whether the parameter Δt represents a maximum busy period of a single task or of multiple task phases. The derivation of the characterization $\gamma_{jy \to i}(\Delta t, \mathcal{Z})$ is subject to Section 6.4.

The main difference to the algorithm for entire tasks is that maximum busy periods have to be computed over multiple task phases, as single task phases of tasks with more than one phase can never be in consecutive execution with themselves, they are always preceded and followed by other phases. This is realized by the variable x' that is used to traverse the different phases of the same task execution, whereas the variable q is used for the consideration of self-delay, to distinguish different executions of the analyzed task (see abstraction level $A^{\langle 2 \rangle}$ in Figure 3).

Another fundamental difference between considering separate task phases and entire tasks is that multiple task phases can be externally enabled, i.e. not all phases are necessarily in consecutive execution once the first phase is externally enabled. This means that a maximum busy period cannot only begin with the first phase, but with each phase accessing a FIFO buffer. This is captured by the outer for-loop in Figure 7 (line 2), which initializes a maximum busy period for all task phases corresponding to actor phases v_{ix} with at least one input edge in E^{ext}. This set is defined as the set of all edges $E^{\langle 2,exp \rangle}$ in the HSDF expansion on level $A^{\langle 2 \rangle}$, but without any self-edges.

Maximum busy periods are initialized under the assumption that the first considered execution $q = 0$ of task phase $\tau_{ix'} = \tau_{ix}$ is externally enabled at $\hat{s}_{ix}^{ext}$ and is not in consecutive execution with its predecessor. Therefore also no previous interference can be excluded, which we capture by defining $\eta_{jy}(0) = 0$ and $\gamma_{jy}(0, \emptyset) = 0$.

After initialization the phases succeeding phase v_{ix} in execution $q = 0$ are traversed by increasing x' and q (lines 10 to 13). It is thereby assumed that all these succeeding phases are in consecutive execution. This implies on the one hand that the maximum busy periods $w_{ix}^{'}$ and w_{ix} are not reinitialized, but extended (line 7). On the other hand, also interference is not considered in separation for each phase, but computed over the entire, extended maximum busy periods and reduced by the interference already considered for the preceding phase executions (lines 5 and 6). For that matter the extensions of the maximum busy periods are populated to form the basis for the next iterations (line 7) and the set $\mathcal{Z}_{ix}$, which contains tuples representing the already considered corresponding actor firings, is extended analogously (line 8).

Finally the maximum finish times of all traversed task phases $\tau_{ix'}$ are recomputed for all considered task executions q (line 9), again based on the assumption that phase τ_{ix} in execution $q = 0$ is externally enabled at $\hat{s}_{ix}^{ext}$ and that all succeeding phases are in consecutive execution.

By construction the maximum finish time computed as $\hat{s}_{ix}^{ext} + w_{ix} - q \cdot P_i$ is a conservative upper bound on the finish time of a task phase $\tau_{ix'}$ if it is in consecutive execution with all its predecessors over $q+1$ executions starting with phase τ_{ix}. Let us now assume that the stop criterion in line 14 were replaced by a simple while (true). Thereby we would consider all phases that can be externally enabled as starting points for maximum busy periods and traverse all phases x' over any $q+1$ executions. This would ensure that all possible cases of consecutive executions are considered for all phases, resulting in periodic upper bounds $\hat{f}_{ix'}$ that hold for any execution. To be more precise, it would hold for a maximum busy period w_{ix} over $q+1$ executions that maximizes $\hat{f}_{ix'}$:

$$\forall_{n \ge 0}: \; f(\iota_{ix'}^n) \le \hat{s}_{ix}^{ext} + (n-q) \cdot P_i + w_{ix} = \hat{f}_{ix'} + n \cdot P_i$$

In the following we present the intuition behind the proof of the validity of the stop criterion in line 14, which is just analogous to the stop criterion used in Figure 6. For a formal proof of the validity of the stop criterion please refer to [13]. The algorithm terminates if it holds for $x = x'$ that $w_{ix}^{'} \le q \cdot P_i$. From this follows for the update of the maximum finish time (line 9) in the next iteration:

$$\hat{s}_{ix}^{ext} + w_{ix} + w_{ix}^{\oplus} - q \cdot P_i \le \hat{s}_{ix}^{ext} + w_{ix}^{\oplus}$$

Furthermore it can be seen that the extension $w_{ix}^{\oplus}$ is always smaller than the w_{ix} computed on initialization. Therefore also the newly computed maximum finish time must be smaller than the one from initialization. As the same reasoning can be applied to all other extensions after the stop criterion is met it follows that the stop criterion must be valid as no maximum finish time computed after the stop criterion is met can be larger than the maximum finish time computed before.

This lets us conclude that the maximum finish times computed with our algorithm are indeed periodic upper bounds on the finish times $f(\iota_{ix}^n)$ of the respective task phase executions, i.e. it holds:

$$\forall_{n \ge 0}: \; f(\iota_{ix}^n) \le \hat{f}(\iota_{ix}^n) = \hat{f}_{ix} + n \cdot P_i$$

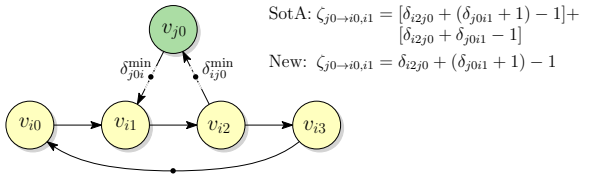

Figure 8: Limiting interference with cyclic data dependencies.

Based on these maximum finish times including interference we derive maximum response times of task phases that we use as firing durations of the corresponding actor phases in the worst-case model in Section 7. Thereby we have to differ between two cases. As can be seen in the expanded version of abstraction level $A^{\langle 2\rangle}$ in Figure 3 the edges from last actor phases to first actor phases are removed in the worst-case model. To maintain temporal conservativeness we have to include the delay from such removed self-edges, i.e. the self-delay on the first phases, in the firing durations of these phases. This is achieved by computing maximum response times for first phases as differences between maximum external enabling and maximum finish times, i.e.:

$$\hat{\rho}_{i0} = \hat{f}(\iota^n_{i0}) - \hat{\varepsilon}^{ext}(\iota^n_{i0}) = \hat{f}_{i0} - \hat{s}^{ext}_{i0}$$

The other actor phases in the worst-case model presented on level $A^{\langle 2\rangle}$ have self-edges coming from their predecessors. This implies that these phases are neither enabled before their maximum external enabling times nor before their maximum internal enabling times, i.e. the maximum finish times of their predecessors. Consequently we compute maximum response times for these phases as:

$$\begin{aligned}\forall_{0<x<\Theta_i}\colon\ \hat{\rho}_{ix} &= \hat{f}(\iota^n_{ix}) - \max(\hat{\varepsilon}^{ext}(\iota^n_{ix}), \hat{f}(\iota^n_{i(x-1)})) \\ &= \hat{f}_{ix} - \max(\hat{s}^{ext}_{ix}, \hat{f}_{i(x-1)})\end{aligned}$$

6.4 Cyclic Data Dependencies

In this section we first explain the derivation of the function $\zeta_{j\to i}(q)$ which is defined in [21] for entire tasks. Then we show that an application of this function for task phases results in a significant overapproximation and subsequently propose a more accurate solution $\zeta_{jy}(\mathcal{Z}_i)$. Afterwards we simplify this solution by exploiting the dependencies between phases of a CSDF actor and finally combine $\zeta_{jy}(\mathcal{Z}_i)$ with $\eta_{jy}(\Delta t)$ to obtain $\gamma_{jy}(\Delta t, \mathcal{Z}_i)$.

In the following we base all our observations on the dependencies between actor phases as shown in the HSDF expansion on abstraction level $A^{\langle 0\rangle}$ in Figure 3. According to Section 3 all dependencies on level $A^{\langle 0\rangle}$ match the data dependencies in the corresponding task graphs one-to-one. This allows us to draw conclusions about interference between task phases based on overlaps of actor phases.

We use the following definitions: Let $\mathcal{P}_{ixjy}$ be the set of all directed paths of edges from an actor phase v_{ix} to an actor phase v_{jy}. Then we define $\delta(\mathcal{P}_{ixjy})$ as the minimum number of initial tokens on any path in $\mathcal{P}_{ixjy}$, with $\delta(\mathcal{P}_{ixjy}) = \infty$ if $\mathcal{P}_{ixjy} = \emptyset$. According to [21] $\delta(\mathcal{P}_{jyix})$ and $\delta(\mathcal{P}_{ixjy})$ can be computed efficiently using the Floyd-Warshall algorithm.

In [21] so-called interference sets containing all firings of an actor v_j that can occur during a firing n of an actor v_i are used to derive $\zeta_{j\to i}(q)$. We redefine these sets for actor phases v_{jy} and v_{ix} instead of actors v_j and v_i as follows:

$$P_{v_{jy}\to\iota^n_{ix}} = \{\iota^m_{jy} \mid n - \delta(\mathcal{P}_{jyix}) < m < n + \delta(\mathcal{P}_{ixjy})\}$$

The intuition behind such interference sets is the following: Consider the HSDF graph depicted in Figure 8. From the semantics of HSDF graphs one can conclude that phase v_{j0} can maximally fire $\delta(\mathcal{P}_{i0j0}) = \delta^{\min}_{ij0}$ times before it must be enabled by an additional token produced by a finished firing of phase v_{i0}. This implies that any firing $m \geq n + \delta(\mathcal{P}_{i0j0})$ of phase v_{j0} cannot be enabled before firing n of phase v_{i0} is finished.

Likewise, phase v_{i0} can maximally fire $\delta(\mathcal{P}_{j0i0}) = \delta^{\min}_{j0i} + 1$ times before it requires an additional token produced by a firing of phase v_{j0}. This implies that any firing $m \leq n - \delta(\mathcal{P}_{j0i0})$ of phase v_{j0} must be finished before firing n of phase v_{i0} is enabled. Negating these constraints then just gives all firings m of a phase v_{j0} that can occur during a firing n of a phase v_{ix}, i.e. all firings in the interference set $P_{v_{j0}\to\iota^n_{i0}}$.

The function $\zeta_{j\to i}(q)$ from [21] is defined as the number of elements in unions of interference sets. By simply replacing tasks with task phases we could redefine $\zeta_{j\to i}(q)$ as:

$$\zeta_{jy\to ix}(q) = |\bigcup_{q'=0}^{q} P_{v_{jy}\to\iota^{n+q'}_{ix}}| = \delta(\mathcal{P}_{ixjy}) + \delta(\mathcal{P}_{jyix}) + q - 1$$

Now reconsider the example depicted in Figure 8. By applying $\zeta_{jy\to ix}(q)$ for a consecutive firing of phases v_{i0} and v_{i1} we obtain $\zeta_{j0\to i0,i1} = |P_{v_{jy}\to\iota^n_{i0}}| + |P_{v_{jy}\to\iota^n_{i1}}|$. This results in the upper equation in the figure in which firings of phase v_{jy} that occur in both $P_{v_{jy}\to\iota^n_{i0}}$ and $P_{v_{jy}\to\iota^n_{i1}}$ are accounted for twice, leading to a significant overapproximation.

To prevent this we consequently redefine $\zeta_{j0\to i0,i1}$ as the number of elements in the union of phase interference sets, i.e. $\zeta_{j0\to i0,i1} = |P_{v_{jy}\to\iota^n_{i0}} \cup P_{v_{jy}\to\iota^n_{i1}}|$, resulting in the lower equation in the figure.

In the following we generalize such interference characterizations for arbitrary sequences of phases.

We define tuples $z = (v_{ix}, q)$ consisting of phases v_{ix} of an actor v_i and a firing index q. For instance, the tuple $(v_{i1}, 0)$ would correspond to the first considered firing of phase v_{i1} and the tuple $(v_{i0}, 1)$ to the second considered firing of phase v_{i0}. An ordering relation on such tuples can be defined as follows: If it holds for two tuples $z = (v_{ix}, q), z' = (v_{ix'}, q')$ that $q < q'$ or if $q = q'$ that $x \leq x'$ then we say that $z \leq z'$. Given this ordering relation we define $\check{\mathcal{Z}}_i = (v_{i\check{x}}, \check{q})$ as the infimum and $\hat{\mathcal{Z}}_i = (v_{i\hat{x}}, \hat{q})$ as the supremum of a set $\mathcal{Z}_i$. Furthermore we consider sets $\mathcal{Z}_i$ containing complete sequences of such tuples, i.e. it holds that:

$$z, z'' \in \mathcal{Z}_i \wedge z < z' < z'' \Rightarrow z' \in \mathcal{Z}_i$$

Using such sets we can now define an interference characterization on such sets as the number of elements in the union of all interference sets of the contained tuples, i.e.:

$$\begin{aligned}\zeta_{jy}(\mathcal{Z}_i) &= |\bigcup_{z\in\mathcal{Z}_i} P_{v_{jy}\to\iota^{n+q}_{ix}}| \qquad (2)\\ &= \max_{z\in\mathcal{Z}_i}(\delta(\mathcal{P}_{ixjy}) + q) + \max_{z\in\mathcal{Z}_i}(\delta(\mathcal{P}_{jyix}) - q) - 1\end{aligned}$$

This function can be further simplified as follows: For two tuples $z, z' \in \mathcal{Z}_i$ with $z < z'$ we can differ between the cases $q < q'$ and $q = q' \wedge x < x'$. According to the HSDF expansion on abstraction level $A^{\langle 0\rangle}$ all phases of an actor lie on a cycle with one token, which implies $\forall_{x,x'}\colon\ |\delta(\mathcal{P}_{ixjy}) - \delta(\mathcal{P}_{ix'jy})| \leq 1$. For the first case it therefore follows that $\delta(\mathcal{P}_{ixjy}) + q \leq \delta(\mathcal{P}_{ix'jy}) + q'$. In the second case we have $x < x'$. As exemplified in Figure 8 for $v_{ix_{out}} = v_{i2}$ we can always find a phase $v_{ix_{out}}$ such that $\forall_{x\leq x_{out}}\colon\ \delta(\mathcal{P}_{ixjy}) = \delta^{\min}_{ijy}$ and $\forall_{x>x_{out}}\colon\ \delta(\mathcal{P}_{ixjy}) = \delta^{\min}_{ijy} + 1$. Consequently it holds that $\delta(\mathcal{P}_{ixjy}) \leq \delta(\mathcal{P}_{ix'jy})$ and we can also conclude for the second case that $\delta(\mathcal{P}_{ixjy}) + q \leq \delta(\mathcal{P}_{ix'jy}) + q'$.

From this finally follows that for any set $\check{\mathcal{Z}}_i$ the first maximum function in Equation 2 is maximal for the supremum $\hat{\mathcal{Z}}_i$ and with an analogous reasoning that the second maximum function is maximal for the infimum $\check{\mathcal{Z}}_i$, i.e.:

$$\zeta_{jy}(\mathcal{Z}_i) = \delta(\mathcal{P}_{i\hat{x}jy}) + \hat{q} + \delta(\mathcal{P}_{jyi\check{x}}) - \check{q} - 1 \qquad (3)$$

Finally note that both $\eta_{jy}(\Delta t)$ and $\zeta_{jy}(\mathcal{Z}_i)$ represent upper bounds on the number of executions of a task phase τ_{jy} that can interfere with the corresponding task phase executions of the firings in $\mathcal{Z}_i$, provided that Δt is an upper bound on the time needed to execute the task phases of τ_i included in $\mathcal{Z}_i$. This allows us to combine the two for a tighter bound on interference as follows:

$$\gamma_{jy}(\Delta t, \mathcal{Z}_i) = \min(\eta_{jy}(\Delta t), \zeta_{jy}(\mathcal{Z}_i))$$

7. BOUNDS ON SCHEDULES AND JITTERS

In this section we present our method to compute bounds on the schedules and upper bounds on the enabling jitters of task phases in step 2 of the analysis flow. In [8] it is shown that temporally conservative upper bounds on enabling jitters can be derived from periodic lower and upper bounds on the enabling times of tasks (or in our case task phases). Following the reasoning in Section 4 we compute these bounds using dataflow models reflecting the best-case and worst-case behavior of an analyzed application.

To determine a lower bound on the enabling times of task phases we make use of the best-case model that is presented on abstraction level $A^{\langle -1 \rangle}$ in Figure 3. According to [8] the start times of actor phases in such a best-case model can be determined using the following Linear Program (LP) (with $E^{\langle -1,exp \rangle}$ the set of all edges on level $A^{\langle -1 \rangle}$ and the firing durations $\check{\rho}_{ix}$ of actor phases v_{ix} assigned to the BCETs of the corresponding task phases):

$$\begin{aligned} &\text{Minimize} \sum_{v_{ix} \in V} \check{s}_{ix} \\ &\text{Subject to } \check{s}_{s0} = 0 \\ &\qquad \forall_{e_{ixjy} \in E^{\langle -1,exp \rangle}} : \ \check{s}_{jy} - \check{s}_{ix} \geq \check{\rho}_{ix} \end{aligned}$$

These start times define a periodic schedule for the actor phases in the best-case model, i.e. it holds that the start time of an actor phase v_{ix} in firing n is equal to $\check{s}_{ix} + n \cdot P_i$. By setting the start time of the first source actor phase v_{s0} to $\check{s}_{s0} = 0$ it holds that all start times are computed relative to the first enabling of the first source actor phase. If we also define the enabling times of task phases relative to the first execution of their strictly periodic source it follows that we can use the start time $\check{s}_{ix}$ of an actor phase v_{ix} in the best-case model to determine a periodic lower bound on the enabling times $\varepsilon(\iota_{ix}^n)$ of a task phase execution ι_{ix}^n, i.e.:

$$\forall_{n \geq 0} : \ \check{\varepsilon}(\iota_{ix}^n) = \check{s}_{ix} + n \cdot P_i \leq \varepsilon(\iota_{ix}^n)$$

To compute upper bounds on the enabling times of task phases in step 2 of the analysis flow we make use of the worst-case model presented as the rolled-back HSDF expansion on abstraction level $A^{\langle 2 \rangle}$. In the first iteration of the analysis flow the firing durations of actor phases are set to the WCETs of the task phases, whereas in subsequent iterations the maximum response times derived in step 3 of the analysis flow are used. According to [8] the start times of actor phases in the worst-case model can be computed by solving the following LP:

$$\begin{aligned} &\text{Minimize} \sum_{v_{ix} \in V} \hat{s}_{ix} \\ &\text{Subject to } \hat{s}_{s0} = 0 \\ &\qquad \forall_{e_{ixjy} \in E^{\langle 2,exp \rangle}} : \ \hat{s}_{jy} - \hat{s}_{ix} \geq \hat{\rho}_{ix} - \delta(e_{ixjy}) \cdot P_j \end{aligned}$$

Note that there is a fundamental difference between the worst-case model used in [8] and the one in this paper. In the model in [8] only edges between different tasks are considered, but no self-edges. For such models the LP determines upper bounds on the external enabling times of tasks. However, as can be seen on abstraction level $A^{\langle 2 \rangle}$, we only remove the self-edge between the last and the first phase of an actor, but the other phases remain connected via self-edges to model internal enablings. This implies that the start times computed for the first phases are upper bounds on the external enabling times of the corresponding task phases and all other start times are upper bounds on total enabling times, i.e. upper bounds on the maximum of external and internal enabling times. However, for the method presented in the previous section we do not only require an upper bound on the external enabling time of the first phases, but of all phases v_{ix} that can be externally enabled, i.e. have input edges e_{jyix} coming from other actors $v_j \neq v_i$. To achieve this we modify the worst-case LP such that external and internal enablings are separated. With $E^{ext} \subset E^{\langle 2,exp \rangle}$ the set of all edges modeling data dependencies between phases of different tasks we formulate the following worst-case LP:

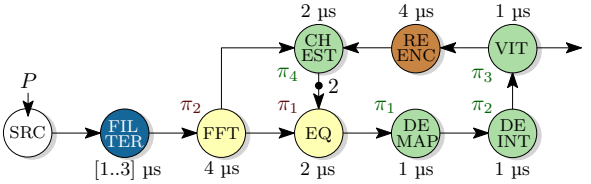

Figure 9: HSDF graph of the packet decoding mode of a WLAN 802.11p transceiver.

$$\begin{aligned} &\text{Minimize} \sum_{v_{ix} \in V} \hat{s}_{ix}^{ext} + \hat{s}_{ix} \\ &\text{Subject to } \hat{s}_{s0} = 0 \\ &\qquad \forall_{e_{ixjy} \in E^{ext}} : \quad \hat{s}_{jy}^{ext} - \hat{s}_{ix} \geq \hat{\rho}_{ix} - \delta(e_{ixjy}) \cdot P_j \\ &\qquad \forall_{e_{ixjy} \in E^{\langle 2,exp \rangle}} : \quad \hat{s}_{jy} - \hat{s}_{ix} \ \geq \hat{\rho}_{ix} - \delta(e_{ixjy}) \cdot P_j \end{aligned}$$

For the first phases of an actor it holds that $\hat{s}_{i0} = \hat{s}_{i0}^{ext}$. With the extended worst-case LP we can bound both external and total enabling times of task phase executions ι_{ix}^n from above as follows:

$$\begin{aligned} \forall_{n \geq 0} : \quad & \varepsilon^{ext}(\iota_{ix}^n) \leq \hat{\varepsilon}^{ext}(\iota_{ix}^n) = \hat{s}_{ix}^{ext} + n \cdot P_i \\ \forall_{0 < x < \theta_i} : \quad \forall_{n \geq 0} : \quad & \varepsilon(\iota_{ix}^n) \quad \leq \hat{\varepsilon}(\iota_{ix}^n) \quad = \hat{s}_{ix} + n \cdot P_i \end{aligned}$$

What is still missing is an upper bound on the total enabling times of the first phases v_{i0} which can be obtained by additionally considering delays from the last phases $v_{i(\Theta_i - 1)}$, i.e.:

$$\begin{aligned} \forall_{n \geq 0} : \ \varepsilon(\iota_{i0}^n) &\leq \hat{\varepsilon}(\iota_{i0}^n) = \max(\hat{\varepsilon}^{ext}(\iota_{i0}^n), \hat{f}(\iota_{i(\Theta_i - 1)}^{n-1})) \\ &= \max(\hat{s}_{i0}, \hat{s}_{i(\Theta_i - 1)} + \hat{\rho}_{i(\Theta_i - 1)} - P_i) + n \cdot P_i \end{aligned}$$

Finally we can formulate an upper bound on the enabling jitter of any task phase execution ι_{ix}^n as the difference between its minimum and maximum enabling times:

$$\begin{aligned} &\forall_{n \geq 0} : \ J(\iota_{ix}^n) \leq \hat{J}_{ix} \text{ with} \qquad (4) \\ &\hat{J}_{ix} = \hat{\varepsilon}(\iota_{ix}^n) - \check{\varepsilon}(\iota_{ix}^n) \\ &\qquad = \begin{cases} \max(\hat{s}_{i0}, \hat{s}_{i(\Theta_i - 1)} + \hat{\rho}_{i(\Theta_i - 1)} - P_i) - \check{s}_{i0} & , \ x = 0 \\ \hat{s}_{ix} - \check{s}_{ix} & , \ \text{else} \end{cases} \end{aligned}$$

8. CASE STUDY

This section demonstrates the benefits of our approach in a case study. We analyze the task graph of a WLAN 802.11p transceiver [1] which is used in safety-critical automotive applications like automated braking systems. A WLAN 802.11p transceiver has several modes and is executed on a multiprocessor system for performance reasons. We only consider the part of the task graph that is active during packet decoding mode. An HSDF model correspond-

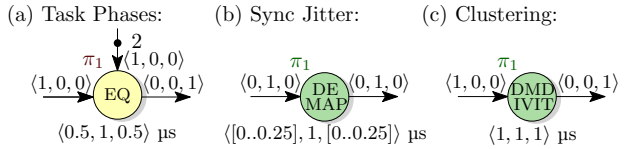

Figure 10: Considered CSDF use-cases for the graph in Figure 9.

	Original		(a) Task Phases				(b) Sync Jitter				(c) Clustering			
P/ µs	17	12	30	20	17	12	24	22	20	16	15	11	10	9
ST+J	34	-	60	-	-	-	40.5	-	-	-	22	-	-	-
ST+C	25	25	40.5	40.5	-	-	31.5	31.5	31.5	-	22	22	-	-
TP+J	34	-	25	25	34	-	40.5	40.5	-	-	18	20	20	-
TP+C	25	25	25	25	25	25	31.5	31.5	31.5	31.5	18	18	18	18

Table 1: End-to-end latencies obtained for cases in Figure 10 (in µs).

ing to the task graph of the packet decoding mode is shown in Figure 9.

A periodic source models the input of this dataflow graph. The source frequency f at which the transceiver operates is typically 125 kHz, which corresponds to a period P of 8 µs. For illustration purposes, however, we vary the source period such that the guaranteed throughput is maximized for the different use-cases. The BCETs and WCETs of the tasks are denoted next to the corresponding dataflow actors.

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).

Our analysis flow allows for the quick verification of different task-to-processor mappings and priority assignments. One such assignment is presented in Figure 9, with different colors indicating different processors and the priorities π_x written next to the tasks (with π_1 the lowest).

In the following we discuss different modifications of the given HSDF graph such that it becomes a CSDF graph and compare the analysis accuracy of our approach considering task phases (TP) to the accuracy of the state-of-the-art approach presented in Section 6.1, which considers task phases as separate tasks (ST). Thereby we further differ between considering interference based on period-and-jitter only (J) and based additionally on cyclic data dependencies (C).

The results for the different cases and approaches are presented in Table 1, with the periods P being the minimum periods for which the approaches do not report a constraint violation. In case no constraint violation occurs the entries in the table give the maximum end-to-end latency from the source to the end of VIT as a measure of accuracy. Moreover, the minimum period defines the maximum guaranteeable throughput, making it another accuracy measure.

Application of the different approaches on the original case depicted in Figure 9 obviously does not show any differences between ST and TP, as each task consist of only one phase. Nevertheless, an improvement of 29 % in minimum periods (and thus throughput) and of 25 % in end-to-end latencies can be observed if cyclic data dependencies are exploited to limit interference (C).

In the first considered modification we assume that we have more information about the tasks than their BCETs and WCET only, such that we know that the read operations of all tasks take the fixed time of 0.5 µs right after the beginnings of task executions and the write operations the same at their ends. This allows us to split all tasks into three phases, as it is exemplified for task EQ in Figure 10(a). Such a modification can in general lead to higher throughput guarantees, smaller required buffer capacities and can be used to relax the requirement from [21] that for a task execution all acquires must happen atomically at its beginning and all releases atomically at its end. For TP we do not observe a difference in throughput or end-to-end latencies compared to the original case. This is due to the feedback loop imposing a rather strict throughput constraint on the tasks in the loop, which does not change by the presented modification as all phases remain "in-the-loop". However, analysis results are significantly worse for ST (up to 76 % lower throughput guarantees and up to 140 % higher end-to-end latencies) which is due to the fact that for ST interference is tripled by considering three phases per task.

The second modification can be used to relax the requirement of atomic synchronization, even if a task cannot be split into phases like in the first use-case. If it can be guaranteed that all acquires happen in the first 0.25 µs and all releases in the last 0.25 µs of each task execution we can model such variances as synchronization jitters, as exemplified for task DEMAP in Figure 10(b). As expected, applying this relaxation on all tasks comes at the cost of worse analysis results in all cases. Moreover we do not see any differences in end-to-end latencies between ST and TP which can be explained by the fact that also with TP a reduction of interference only occurs on the "out-of-the-loop" phases one and three, whereas the externally enabled "in-the-loop" phases have maximal interference. However, throughput guarantees are better for TP than for ST (up to 20% higher) as for small periods the cycles over the three phases become more critical than the feedback loop cycle due to the overestimation of interference in ST.

The third modification is a technique called task clustering [4] which can be used to remove synchronization overhead, as well as any interference between clustered tasks, leading to potentially better analysis results. We apply this technique on the tasks DEMAP, DEINT and VIT, resulting in the task cluster DMDIVIT depicted in Figure 10(c). Compared to the original case we see indeed an improvement in both throughput and end-to-end latencies for both ST and TP. For ST the improvement is moderate (up to 13 % higher throughput guarantees and up to 35 % lower end-to-end latencies compared to the original case) as interference of CHEST is still accounted for all three subtasks. For TP, however, only the subtask DEMAP experiences maximal interference, which leads to a more significant improvement (up to 25 % higher throughput guarantees and up to 41 % lower end-to-end latencies).

This allows us to conclude that the introduction of our maximum response time computation considering multiple task phases results in a significant accuracy improvement of state-of-the-art for practically relevant use-cases. On top of that we observe that the explicit consideration of cyclic data dependencies for limiting interference consistently leads to even better results. Finally note that we have verified the temporal conservativeness of our results using a high-level simulator called HAPI [15] and that all analysis run-times were in the microsecond range on a standard PC, showing the applicability of our approach for a quick verification of temporal constraints.

9. RELATED WORK

In [8] a combination of dataflow analysis with real-time analysis techniques is presented which can analyze single-rate streaming applications that are executed on multiprocessor systems with processor sharing and SPP scheduling.

This approach is extended in [21] by the consideration of the effect that cyclic data dependencies limit interference, resulting in an increased analysis accuracy. In [22] the introduction of an iterative buffer sizing enables the exploitation of a trade-off between interference and pipeline parallelism, leading to both smaller buffer capacities and higher throughput guarantees. Lastly, the approach in [14] further increases accuracy by replacing the period-and-jitter characterization used in [22] by a combination of offsets and an explicit consideration of data dependencies between interfering tasks.

The limitation of the above approaches to single-rate applications that are expressible with HSDF graphs is removed in [9], considering different, but fixed rates on the inputs and outputs of tasks, which allows to analyze applications expressible with Synchronous Dataflow (SDF) graphs. The main contribution of [9] is an interference characterization which captures bursts due to rate conversions more accurately than period-and-jitter. The maximum response time computation, however, is unchanged, making our contributions on the computation of maximum response times over multiple phases orthogonal to the ones from [9]. In contrast to all these works our approach can analyze any applications expressible with CSDF models, allowing the analysis of tasks with cyclo-statically changing rates and execution times. This is also relevant for applications that are naturally expressible with HSDF or SDF graphs, as CSDF models can be used to relax the requirement of atomic synchronization on the boundaries of task executions.

The analysis approaches in [23], [12] and [16], that is part of the MAST framework [5], combine offsets with a notion of precedence constraints, which allows for task chains to exclude interference of succeeding tasks if it is already considered for preceding tasks. However, an explicit notion of task phases does not exist, such that maximum busy periods are still computed on a per-task basis. This prevents an interference reduction for multiple task phase executions, i.e. $q > 0$. The approaches are further limited to single-rate applications and acyclic task graphs, such that neither applications with feedback loops nor with limited buffer capacities can be correctly analyzed and also the effect that cyclic data dependencies limit interference cannot be exploited.

Finally, the approach in [17] computes maximum response times of tasks consisting of multiple phases. While the approach allows for different priorities for different phases, as well as pipelining between different executions of the same task, it is limited to chains of single-rate phases with only one input at the beginning and one output at the end. This implies that all tasks must be structured like the CSDF actor presented in Figure 10(c), while our approach supports arbitrary multi-phase tasks with different rates. Moreover [17] is based on the SymTA/S approach [11] and as such uses traffic propagation to model data dependencies between different tasks. This prevents to maintain the correlation between events, as well as a consideration of arbitrary cyclic data dependencies. Our approach uses periodic bounds on schedules obtained by dataflow analysis to consider data dependencies, which captures the correlation between events correctly, allows to model both feedback loops and FIFO buffers and enables an exploitation of the effect that cyclic data dependencies limit interference, which results in a significantly higher analysis accuracy.

10. CONCLUSION

This paper presented a temporal analysis approach that is capable of giving design-time guarantees on the throughput and latency of cyclic real-time stream processing applications that can be expressed with CSDF models and that are executed on multiprocessor systems with processor sharing and SPP scheduling. The approach addresses two major shortcomings of existing techniques, which are the limitation to the accurate analysis of single-rate applications and the requirement that task synchronization must occur atomically at the boundaries of tasks. For that purpose a novel analysis technique for tasks consisting of multiple phases was introduced that prevents to account for the same interference multiple times. Moreover a multi-phase interference characterization was presented which exploits that cyclic data dependencies limit interference between tasks, resulting in a significantly higher analysis accuracy.

In a case study the presented approach was applied for different use-cases of CSDF modeling and subsequently evaluated with respect to applicability, efficiency and accuracy. Future work includes the development of an iterative buffer sizing technique similar to the one in [22], but applicable for applications expressible with CSDF models.

11. REFERENCES

[1] P. Alexander et al. Outdoor mobile broadband access with 802.11. *IEEE Communications Magazine*, 45(11):108–114, 2007.

[2] G. Bilsen et al. Cycle-static dataflow. *IEEE Trans. on Signal Processing*, 44(2):397–408, 1996.

[3] G. Buttazzo. *Hard real-time computing systems*. Springer US, 2011.

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

[5] M. Gonzalez Harbour et al. MAST: Modeling and analysis suite for real time applications. In *ECRTS*, pages 125–134, 2001.

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

[7] J. Hausmans et al. Resynchronization of cyclo-static dataïñĆow graphs. 2011.

[8] J. Hausmans et al. Dataflow analysis for multiprocessor systems with non-starvation-free schedulers. In *SCOPES*, pages 13–22, 2013.

[9] J. Hausmans et al. Temporal analysis flow based on an enabling rate characterization for multi-rate applications executed on MPSoCs with non-starvation-free schedulers. In *SCOPES*, pages 108–117, 2014.

[10] J. Hausmans et al. A refinement theory for timed dataflow analysis with support for reordering. In *EMSOFT*, 2016.

[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] P. Kurtin et al. Appendix to temporal analysis of static priority preemptive scheduled cyclic streaming applications using CSDF models. Technical report, Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, 2016.

[14] P. Kurtin et al. Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications. In *RTAS*, pages 1–12, 2016.

[15] P. Kurtin et al. HAPI: An event-driven simulator for real-time multiprocessor systems. In *SCOPES*, pages 60–66, 2016.

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

[17] J. Schlatow et al. Response-time analysis for task chains in communicating threads. In *RTAS*, pages 1–10, 2016.

[18] S. Stuijk et al. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *DAC*, pages 777–782, 2007.

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

[20] M. Wiggers et al. Simultaneous budget and buffer size computation for throughput-constrained task graphs. In *DATE*, pages 1669–1672, 2010.

[21] P. Wilmanns et al. Accuracy improvement of dataflow analysis for cyclic stream processing applications scheduled by static priority preemptive schedulers. In *DSD*, pages 9–18, 2014.

[22] P. Wilmanns et al. Buffer sizing to reduce interference and increase throughput of real-time stream processing applications. In *ISORC*, pages 9–18, 2015.

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