Compositional Temporal Analysis Method for Fixed Priority Pre-emptive Scheduled Modal Stream Processing Applications

Authors omitted for blind review

ABSTRACT

Modal real-time stream processing applications often contain cyclic dependencies and are typically executed on multiprocessor systems with processor sharing. Most real-time operating system kernels for these systems support Static Priority Pre-emptive (SPP) scheduling, however there is currently no suitable temporal analysis technique available for this type of systems.

In this paper we present a compositional temporal analysis approach for modal and cyclic stream processing applications executed on SPP scheduled multiprocessor systems. In this approach locks and barriers are added such that the temporal behavior of modes can be characterized independently. As a result, the composition of modes does not change their characterization. This enables the use of an existing Structured Variable-Rate Phased Dataflow (SVPDF) model based dataflow analysis technique to determine the worst-case temporal behavior. The SVPDF model and the parallel implementation including locks and barriers are generated by a multiprocessor compiler.

The applicability of the analysis approach is demonstrated using a WLAN 802.11p application. Conditions under which pipelined execution can be achieved are identified. The analysis results are verified with a dataflow simulator that supports sharing of resources.

CCS Concepts

•Software and its engineering → Formal software verification;

Keywords

Real-time, dataflow, multiprocessor systems, modal systems, automatic parallelization

1. INTRODUCTION

Stream processing applications such as software defined radios and vision applications are typically executed on embedded multiprocessor systems under real-time constraints. These applications often contain multiple processing modes and cyclic dependencies. Examples of processing modes found in software defined radios are the detection, synchronization and decoding mode. The cyclic dependencies are a result of feedback loops and the use of bounded FIFO buffers for inter-task communication.

Stream processing applications are described as task graphs of which the tasks are executed on shared processors. The tasks executed on a processor are scheduled according to a scheduling policy. Examples of scheduling policies are Time Division Multiplex (TDM), Round-Robin (RR) and SPP.

The minimum throughput of a modal stream processing application can be determined using dataflow analysis techniques given a Variable-Rate Phased Dataflow (VPDF) [17] or a Finite State Machine-based Scenario-Aware DataFlow (FSM-SADF) [2] model if budget schedulers are applied [15, 16]. An example of a budget scheduler is TDM. For these budget schedulers it is possible to derive the worst-case response time of each task independently of the schedule of the other tasks, after which a compositional temporal analysis method can be applied. However, many embedded operating systems only support the SPP scheduling policy. Dataflow analysis techniques for systems that use SPP have recently been introduced for multi-rate applications that can be modeled as SDF graphs [6]. However, there is currently no temporal analysis technique for modal stream processing applications that contain cycles and that are executed on multiprocessor systems which make use of SPP task scheduling.

In this paper we present a compositional temporal analysis approach for modal stream processing applications executed on multiprocessor systems using SPP task scheduling per processor. A compiler inserts locks and barriers in the application such that the temporal behavior of each application mode can be characterized in isolation. The locks ensure that tasks belonging to different modes do not interfere, and the barriers make the response times of the tasks independent of the production moments of tasks that belong to other modes. The locks and barriers result in additional constraints that are included in a hierarchical SVPDF model of the application. This model is used to verify the satisfaction of the throughput constraint and to compute the required buffer sizes by recursively applying a recently introduced dataflow analysis technique. Furthermore, it is shown that the approach supports response times of tasks larger than the period of the source, and allows the use of budget scheduling besides SPP scheduling. The applicabil-

SCOPES '16 May 23–26, 2016, Sankt Goar, Germany

DOI: 10.475/123_4

ity of the approach is demonstrated using a WLANp application. This application can be executed in a pipelined fashion despite the additional constraints introduced by the locks and barriers.

The outline of this paper is as follows. We first discuss related work in Section 2. The basic idea behind our analysis approach is presented in Section 3. Section 4 describes the analysis flow we use to analyze modal applications. The SVPDF model that we use is described in Section 5. In order to be able to verify if a periodic source can execute strictly periodically when switching between modes, additional constraints are introduced into the SVPDF model as described in Section 6. The response time equations that are introduced in Section 7 can be used to apply the presented analysis approach to systems in which also other scheduling policies are applied than SPP. Conditions under which response times larger than the source period are allowed are stated in Section 8. The applicability of the analysis approach is demonstrated in Section 9. The conclusions are stated in Section 10.

2. RELATED WORK

Mode changes in SPP scheduled single processor systems have been studied extensively [5,12,13]. These works present analysis techniques to determine whether deadline misses can occur during a mode change. Overloads can be prevented by the schedulers by delaying the release of tasks [11]. A limitation of these techniques is that the next mode transition may only be started after the previous mode transition is completed. Furthermore, these techniques are only applicable for acyclic task graphs.

Dataflow models are applicable for cyclic task graphs, and a number of dynamic dataflow models have been developed that are also suitable for modal stream processing applications. In the FSM-SADF [2] dataflow model modes are described as scenarios where the possible transitions between scenarios are encoded in a non-deterministic finite-state-machine. Processor sharing can be supported if schedulers are applied that belong to the class of budget schedulers to which TDM schedulers belong.

In this paper we use the SVPDF [4] model that allows a hierarchical description of nested modes in an application. The dependencies are explicit in this model and it has been shown that a deadlock-free model and implementation can be automatically derived from a sequential description of an application in the OIL language [3]. The SVPDF model has only been used in combination with budget schedulers. In this paper, however, we will show that the SVPDF model can be used if SPP schedulers are applied.

Throughput analysis of systems with multiple applications that are modeled as Mode-Controlled Dataflow (MCDF) and scheduled by SPP has been presented in [10]. However, this approach only addresses interference between tasks that belong to different applications, while in this paper we derive the interference between tasks of the same application.

Only recently, a throughput analysis has been introduced in [7] for multi-rate applications that are modeled by Synchronous Dataflow (SDF) graphs and that are executed on multiprocessor systems using SPP schedulers. By introducing an enabling rate characterization in [6], the accuracy of the analysis technique is improved. The analysis flow based on the enabling rate characterization is further improved in [18] by taking into account that cyclic dependencies limit the maximum interference between tasks. We make use of this observation in this paper, since we introduce cycles to make the execution of tasks mutually exclusive.

Locks have been introduced in modal applications to make tasks scheduled by budget schedulers mutually exclusive, which can improve the accuracy of dataflow analysis [8]. In this paper we generalize this approach for budget schedulers, making it applicable for systems with SPP schedulers.

3. BASIC IDEA

In this section we use a didactic example to explain the basic idea behind our compositional temporal analysis approach. This approach is suitable for modal stream processing applications that are executed on multiprocessor systems using SPP task scheduling per processor. The periodic source in these applications imposes a throughput constraint.

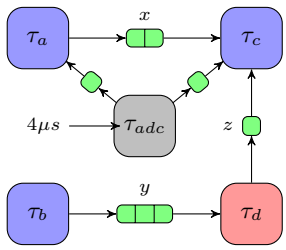
The task graph used in our didactic example is derived by a multiprocessor compiler from the OIL program that is shown in Figure 1a. An OIL program is a kind of C-program with some adaptations that facilitate automatic parallelization. The OIL program contains two potentially endlessly iterating while-loops, where each while-loop corresponds to a mode. After parallelization, the task graph in Figure 1b is obtained. Each task in the graph corresponds to a function in the OIL program, e.g. τ_a corresponds to function a. The color of a task represents the mapping to a processor. Each FIFO buffer in the task graph corresponds to a variable that is communicated between two functions. The source task has two output buffers such that data can be sent to the tasks that belong to an active mode.

Besides a task graph, also an SVPDF dataflow model is created by our compiler. This model contains nested blocks where each block corresponds to a while-loop in the OIL program. These blocks are depicted as dashed blocks in the SVPDF model. The actors in a block correspond with the functions inside the while-loop. The SVPDF model that corresponds to our didactic OIL program is shown in Figure 1c. The nodes on the boundaries of the dashed blocks are called port actors. The port actors at the inputs of a block perform up-sampling, i.e. multiply the number of tokens with a factor that is equal to the number of iterations of the while-loop. The port actors at the outputs of a block perform down-sampling. Port actors have by definition a firing duration equal to zero. The other actors have a firing duration that corresponds to the response times of the tasks. Derivation of these response times is discussed in subsequent paragraphs. The solid black edges between the actors in the SVPDF model denote dependencies between actor firings. The use of FIFO buffers with a bounded capacity in the task graph results in cyclic dependencies in the SVPDF model. The number of tokens on a cycle in the SVPDF model corresponds to the capacity of a buffer. The reason for the inclusion of the other edges in the SVPDF model will be explained in subsequent paragraphs.

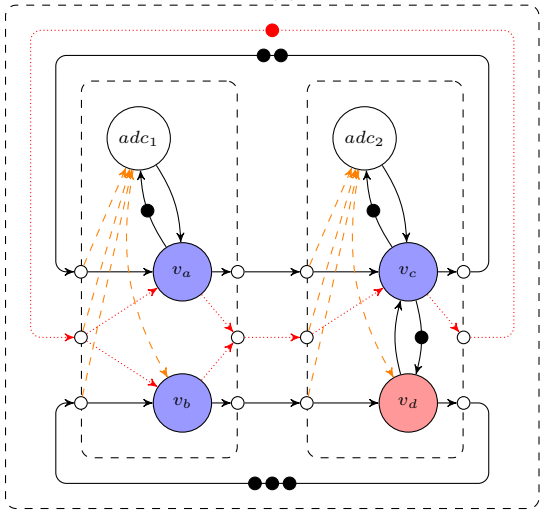
The SVPDF model is the input of our compositional temporal analysis method which is described in Section 4. This method requires that response times of tasks can be determined independently of when tasks in other modes execute. Given that the response times are independent then the firing durations of the actors in one mode is also independent of the firing durations of the actors in another mode. Because the firing durations are independent we can analyze

```
source ADC @ 250 kHz;
loop{
  loop{
    x=a(ADC);
    y=b();
  }while(..);
  loop{
    z=d(y);
    c(ADC, x, z);
  }while(..);
} while(1);
```

(a) Modal OIL program



(b) Corresponding task graph



(c) SVPDF mode of Figure 1a including additional constraints of a lock and barriers

Figure 1: Example of a modal program and the corresponding task graph and dataflow model

whether each deepest nested block fulfills the constraints imposed by the source independently of other blocks. After this is confirmed, we can ascend the hierarchy of blocks and perform the same analysis on the next hierarchical level of the SVPDF model.

The fact that response times of tasks in one mode can be analyzed independently of tasks in another mode is achieved by making use of locks and barriers. Locks prevent that tasks belonging to different modes can execute at the same moment in time. Barriers are used to verify that all inputs of a mode are available before the periodic source in a mode finishes its first execution. As a result, the response times can be determined relative to the executions of the periodic source in a block and become independent of the production moments of tasks that belong to other modes.

Dataflow analysis for systems that make use of non-starvation-free schedulers such as SPP has been described in [6,7]. These approaches make use of a slightly modified version of the Worst-Case Response Time (WCRT) equation that has been proposed by Tindell [14]. This response time equation computes a so-called busy period which calculates the maximum time it takes to finish q executions of τ_i and is defined as:

$$w_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{P_j} \right\rceil \cdot C_j \qquad (1)$$

where J_j is the jitter in the so-called external enabling time

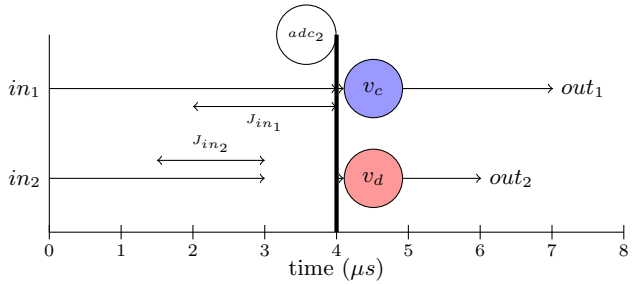


Figure 2: Barrier at $t = 4\mu s$ guarantees that no task belonging to a mode can start before all inputs are available and the source of that mode finishes its firing

of τ_j and $hp(i)$ the set of tasks executing on the same processor with a priority higher than τ_i. The external enabling time of a task is defined as the time at which the task can read sufficient locations from the adjacent buffers. Only values of q for which it holds that $w_i(q) \geq q \cdot P_i$ need to be considered in (1) according to [14].

According to [6,7] is the worst-case response time relative to the external enabling time of a task equal to:

$$\hat{R}_i = \max_{1 \leq q} (w_i(q) - (q-1) \cdot P_i) \qquad (2)$$

From (1) and (2) we conclude that the WCRT of τ_i depends on the jitter J_j of the inputs of the higher priority tasks on the same processor. These inputs can be produced by tasks in a different mode. As a result, the WCRT of τ_i cannot be determined independently of the tasks in other modes, because the jitter can be a result of the jitter in the finish times of tasks in other modes. This is even the case if the execution of these tasks has been made mutually exclusive by making use of locks. Therefore, besides the locks, also barriers are needed to make the jitter of tasks in a mode independent of the production moments of tasks that belong to other modes.

Barriers are used to guarantee that none of the tasks in a mode can start before all inputs for that mode are available and the source inside the mode has finished its firing. Therefore, the start times of v_c and v_d in Figure 2 are only related to the finish time of the source actor in the mode, adc_2, and not to the arrival times of the input data. As a result, the jitter in the production moments of tasks outside a mode has no influence on the response time of the tasks that belong to the mode.

A barrier in the implementation is modeled with additional dependencies in the SVPDF model. These dependencies are depicted as orange dashed arrows in Figure 1c. Moreover, the locks which make the execution of tasks mutually exclusive are modeled with the red dotted edges in Figure 1c. These edges in combination with a single token on the cycles created by these edges guarantee that actors belonging to different blocks, and thus modes, can not fire simultaneously.

4. ANALYSIS FLOW

In this section the applied temporal analysis flow is presented. In this flow, blocks in the SVPDF model are analyzed recursively, starting at the deepest nested blocks in the model, and subsequently one level upwards at a time after flattening the blocks of a hierarchical level. The temporal

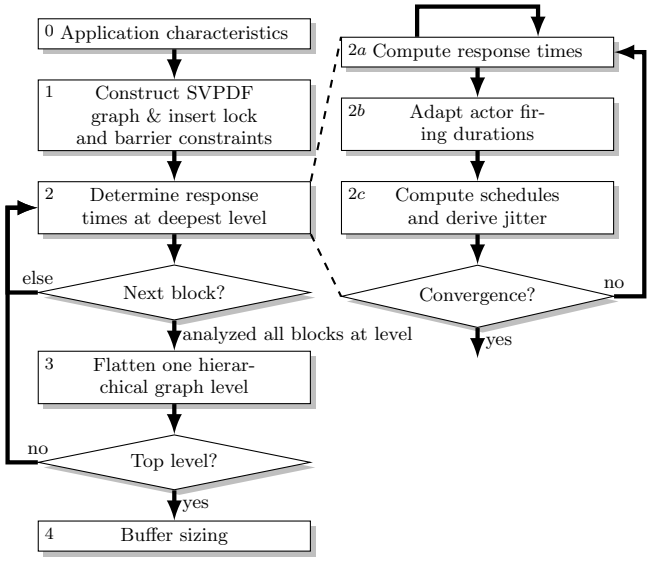


Figure 3: Overview of the analysis flow for modal applications

analysis flow incorporates the flow presented in [18] for the computation of the worst-case response times of the tasks. We present first an overview of the steps in the analysis flow after which the flattening step in the flow is explained in more detail in Section 4.1.

The applied analysis flow is shown in Figure 3. This flow is used to verify whether the temporal constraint imposed by the periodic source (sink) in the blocks can be satisfied. To verify these constraints the worst-case response times of the tasks are computed. The last step in the flow computes sufficiently large buffer capacities.

The analysis flow in Figure 3 contains five steps of which some of the steps are executed repeatedly. Step 2 consists internally of three steps which are executed repeatedly. In step 0, the input information for the analysis flow is gathered. This is the information about the topology of the task graph, the task-to-processor assignment, the applied task scheduling policy for each processor including the scheduling parameters such a priorities, worst- and best-case execution times of the tasks, and the temporal constraints imposed by the periodic source and/or sink inside a block.

Based on the nesting of the functions in an OIL program, an SVPDF model is generated in step 1. Locks and barriers are inserted to enable compositional analysis of blocks as discussed in Section 3. As a consequence, the response times of tasks belonging to a mode can be determined by analyzing each mode in isolation.

In step 2a of the flow, a lower and an upper bound on the response time of the tasks in a block at the deepest hierarchical level is computed. This is done under the assumption that the inputs of the block have arrived before the source of the task has finished its execution, which is verified later. These response times are used to update the minimum and maximum firing durations of the actors in the SVPDF model in step 2b. Two periodic schedules are computed using these firing durations, which bound the start times of the actors. From these start times, the jitter in the production moments of the actors is computed. Given these jitters the response times are recomputed in step 2a, 2b, and 2c, which are repeated until the jitters and response times remain unchanged, or exceed a predefined limit. Step 2 is repeated until all blocks at the deepest hierarchical level have been analyzed.

Once the response times of all the tasks at the deepest hierarchical level have been computed the modes at that level are flattened in step 3. The flattening step is described in more detail in the next section.

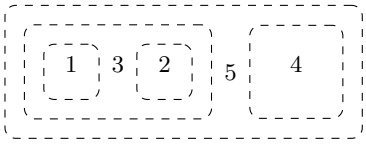


Figure 4: An SVPDF model with nested blocks

Step 2 and 3 are repeated until all levels in the hierarchy of the application are flattened and the top level is reached, or a violation of the temporal constraints is detected. The order in which modes are flattened is explained using Figure 4 which shows the nested block structure of an SVPDF model. In the first iteration of the analysis flow the modes at the deepest level are analyzed, i.e., mode 1 and 2. After these modes are flattened, mode 3 and 4 are analyzed at the next hierarchical level. Finally, mode 5 can be analyzed after all other modes have been flattened.

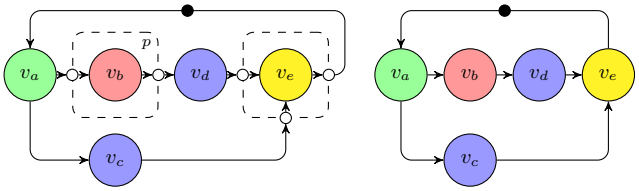
In the final step of the flow, the buffer capacities are determined, given the computed best-case and worst-case schedules for the actors in each of the blocks.

4.1 Flattening of a hierarchical level

The flattening step of the analysis flow removes the hierarchical boundaries of the blocks of one hierarchical level in an SVPDF model. This step is performed after the response times of the actors in the blocks at one hierarchical level have been derived using the analysis flow in Figure 3. In the remainder of this section we will first describe how a hierarchical level can be flattened. Next, we introduce the constraints on the flattening step imposed by the periodic source. Finally, we will discuss the consequences of a flattened block on the jitter and response time of tasks at a higher hierarchical level.

The WCRT of tasks can be calculated under the assumption that tasks in a block execute an infinite number of times. This is possible because the WCRT equation as stated in (2) assumes an infinite number of executions of strictly periodically executing tasks. However, tasks in a block do not execute after a switch to a different block. Therefore the tasks do not execute infinitely often, however, tasks that do not execute, do not cause interference, which reduces the response time. The actual number of times a task executes depends on the value of a so-called parameter [3] of a block, which indicates how often a block will execute. The parameter can range from one to infinite and is only a modeling construct without a counterpart in reality. Therefore an upper bound on the actual response time of the tasks in a block can be calculated by assuming that tasks in a block execute infinitely often.

The structure in an SVPDF model allows blocks to be flattened. A block is flattened by setting the response times of tasks inside the block to the WCRT calculated under the worst-case assumption that tasks in a block execute infinitely often. Therefore, the WCRT is valid for every pos-



(a) Hierarchical SVPDF model (b) HSDF model

Figure 5: Flatting an example of an SVPDF model into a HSDF model

sible parameter value. As a result, only a single iteration of the block needs to be considered at higher hierarchical levels. In the SVPDF model a single iteration of a block is equal to a parameter value of one of the corresponding blocks. A parameter value of one flattens the block, since the boundaries of a block and the port actors become redundant because up- or down-sampling of tokens is not needed anymore. Therefore, the SVPDF model in Figure 5a can be flattened into the Homogeneous Synchronous Dataflow (HSDF) model shown in Figure 5b. The flattened model can be analyzed using the approach presented in [18], which is applied in step 2 of our flow.

A flattened block should still adhere to the constraints imposed by the periodic source inside the block. The constraints are a result of the source task in an application that must be able to execute strictly periodically independent of the block that is active. Two constraints must therefore be satisfied [3]. The first one states that when the same block is executed repeatedly the source actor in that block should execute periodically. The other constraint that must hold is that the source in the block after a mode switch must execute exactly one period after the last execution of the source in the previous block. The first constraint is fulfilled by enforcing that a strict periodic schedule is computed for the source actor in a block. The second constraint is verified in the analysis flow, as will be discussed in Section 6. Satisfaction of both constraints indicate that a schedule is valid for all parameter values.

Now, it might appear that flattening a block is not allowed because the production moments of tasks in nested blocks seem to depend on the number of iterations a block is executed. For example, in Figure 5a the enabling of v_d depends on the finish time of the p'th execution of v_b. The enabling jitter of v_d therefore appears to depend on the block parameter value p which would prevent the block to be flattened. However, we can use the fact that the computed schedules of the actors in a block are periodic. The actual start time of an execution of a task τ_i will be in between the schedule that forms a lower bound, $\check{s}_i$, and the periodic schedule that is used as an upper bound, $\hat{s}_i$. Therefore we can conclude from (6) that the enabling jitter of v_b is independent of p.

$$J_d(p) = \hat{s}_d - \check{s}_d \tag{3}$$
$$= (\hat{s}_b(p) + \hat{\rho}_b) - (\check{s}_d(p) + \check{\rho}_b) \tag{4}$$
$$= \hat{s}_b + (p-1)\cdot P_b - \check{s}_d + (p-1)\cdot P_b + \hat{\rho}_b - \check{\rho}_b \tag{5}$$
$$= \hat{s}_b - \check{s}_d + \hat{\rho}_b - \check{\rho}_b \tag{6}$$

In the equations above $J_b(p)$ is the enabling jitter of v_b for iteration p, $\hat{s}_b$ the latest possible enabling time of v_b in the periodic schedule and $\check{s}_b$ the earliest possible enabling time. The jitter caused by τ_b itself depends on its maximum firing duration $\hat{\rho}_b$ and minumum firing duration $\check{\rho}_b$. Because the enabling jitter for v_b is independent of p, we can use a single iteration of a block for the calculation of the enabling jitter of actors at a higher hierarchical level.

The response time of tasks at a higher hierarchical level can be calculated by setting the parameters of the nested blocks to one, because this results in the maximum interference for tasks at a higher level. The response time for the tasks at a higher hierarchical level is determined by the response time, period, and jitter of tasks at that level according to (2) after additional constraints are added to ensure that blocks can be analyzed in isolation. The jitter caused by tasks in nested blocks is independent of the parameter value of that block, according to (6) as explained in the previous paragraph. Increasing the parameter value of a block changes the period of the tasks at a higher hierarchical level. Executing a nested block more often results in a less frequent execution of the tasks at a higher hierarchical level. The period of these tasks is thus effectively increased by executing nested blocks more often. An increase of the period of tasks results in lower worst-case response time according to (2). The worst-case response times are therefore also independent of the parameter value of a block. Therefore, the minimal parameter value of one can be used for nested blocks to calculate an upper bound of the response time of tasks at higher hierarchical levels.

As an example, consider flattening of the nested block in the SVPDF model in Figure 5a, such that the response time of the tasks at the highest level in the model can be determined. First, the response times of tasks τ_b and τ_e at the deepest nested level are determined by analyzing their blocks. The blocks are flattened by fixing the response times of these two tasks to the computed worst-case response times, and removing the boundaries of the blocks as shown in Figure 5b. Using fixed values for the response times of the tasks in nested blocks is allowed since the additional constraints that are a result of the added locks and barriers ensure that tasks outside a block cannot increase the response time of tasks inside a block. After the nested block are flattened, the response times of τ_a, τ_c and τ_d can be calculated.

Flattening of a hierarchical level removes some potentially useful information. Consider again the SVPDF model shown in Figure 5a where the colors represent the mapping to a processor. On one processor, we have that v_c has a lower priority then v_d. The time at which v_d becomes enabled is dependent on p. From a certain value of p, v_c will always finish before v_d is enabled and therefore v_d will not interfere with v_c. The knowledge about a minimum number of block executions can be used as offset information to obtain a less pessimistic response time for v_c. The use of offset information to obtain more accurate results for cyclic applications that do not contain modes is presented in [9].

5. SVPDF MODEL

In this section we present the SVPDF model. The SVPDF model is based on the VPDF model but contains additional structure in the form of hierarchical blocks. These blocks can be seen as `do-while` loops that iterate for an unknown number of times. In Section 7.1 properties of the model are used to prove that tasks in different blocks execute mutually exclusive when locks are applied.

An SVPDF model is a directed graph $G = (V, E, P, \delta, \rho)$

that consists of a set of actors V connected by a set of directed edges E. An actor $v_i \in V$ communicates to another actor v_j by producing tokens on an edge $e_{ij} \in E$. Initially there are $\delta(e_{ij})$ (δ_{ij} in short) tokens on an edge. An actor v_i is enabled to fire if there are tokens on all its incoming edges. After its firing duration ρ_i a token is produced on all its outgoing edges. The number of tokens on a cycle remains constant and is for example used to model the capacity of a First-In-First-Out (FIFO) buffer.

Hierarchy is introduced in the form of blocks with port actors on the block boundaries. Actors inside a block fire $p \in P$ times compared to actors one level higher in the hierarchy of blocks. The value of p is unknown during analysis and can be infinite. Communication between actors inside and outside a block can only be through the port actors. These port actors perform an up- or down-sampling of tokens such that a single token coming from outside the block is replicated p times, and vice versa for communication in the opposite direction.

The blocks are used to describe modal behavior of an application, where it is unknown how many iterations an application remain in a certain mode. Tasks in these modes can interact with the environment via periodic sources and sinks. Only tasks in the active mode can read from a source or write to a sink, such that every sample produced by a source is only read in a single mode. Therefore an actor derived from a source or sink is copied into every block in which it is used. Such an actor only produces or consumes tokens if the mode corresponding to the block is active. The implementation and analysis of periodic sources is discussed further in Section 6. These sources and sinks execute periodically and therefore impose a throughput constraint on the execution of the application.

6. PERIODIC SOURCE CONSTRAINTS

In this section we derive the conditions that need to be satisfied for applications containing multiple modes in which the same source is accessed. Additional constraints need to be introduced to verify that the source task can execute strictly periodically when switching between modes.

The source task in an application needs to execute strictly periodically. Therefore, the N actors derived from the source task, for all N blocks where the source is read, need to fire periodically. The source actors fire periodically when the time between the start times of firings of the same, or consecutive source actors, is exactly one period P of the source task. The two additional constraints in (7) and (8) on the start times of the source actors, v_s^q with $q \in \{0, 1, \ldots, N-1\}$, are therefore added.

$$\hat{s}_s^q(i+1) = \hat{s}_s^q(i) + P \tag{7}$$

$$\hat{s}_s^{(q+1) \bmod N}(i+1) = \hat{s}_s^q(i) + P \tag{8}$$

where $\hat{s}_s^q(i)$ is the upper bound on the start time of source actor v_s^q in iteration $i \in \mathbb{N}$ if that source actor fires in iteration i. The constraint in (8) is added in step 2c of the analysis flow presented in Section 4. This constraint is added in the hierarchical level where all the blocks containing source actors are flattened such that each source actor fires only once before switching to the next source actor.

As a result of the added constraints the source actors must fire strictly periodically. A delay in the enabling time of a source actor will immediately lead to a violation of the throughput constraint in the block one level higher in the hierarchy than the blocks containing the source actors, which is $2P$ in the example since there are two source actors. A delay in the enabling of a source actor occurs when the inputs of a mode arrive too late and the constraints resulting from a barrier do not enable the source actor in time. A violation of the throughput constraint occurs when the sum of the response times of the tasks on the path of edges without tokens from one source actor to the next source actor, via the constraint that result of locks and barriers, is larger than one period. An example of such a path is seen in Figure 1c from actor src_2 to v_d, v_c via the red dotted and orange dashed edges originating from a lock and barrier to src_1. The combination of the constraints of the strictly periodic executing source and the barriers and locks ensures that mode transitions can be analyzed.

7. RESPONSE TIMES

In the previous sections, constraints are enforced to enable independent analysis of modes. In this section a general WCRT equation is derived that makes use of the fact that modes can be analyzed in isolation after additional constraints are added in the dataflow model. The equation is valid for schedulers in the non-starvation-free class which includes SPP, RR and TDM schedulers. We make use of the prove that the constraints resulting from locks prevent any interference of tasks in other modes as is given in Section 7.1.

In general, the WCRT $\hat{R}_i$ of a task τ_i consists of its Worst-Case Execution Time (WCET) C_i and an upper bound on the interference I of tasks assigned to the same processor, as shown in the following equation:

$$\hat{R}_i = C_i + I_{scheduler}(i, \hat{R}_i) \tag{9}$$

The interference depends on the type of scheduler used on a processor. We will first consider SPP schedulers and limit $\hat{R}_i$ to P_i, to satisfy the constraints presented in Section 6. Since this limitation implies that $w_i \leq P$, only a single execution of τ_i needs to be considered such that $q = 1$. The WCRT of (2) can then be simplified to the following equation for the interference on τ_i:

$$I_{SPP}(i, \Delta t) = \sum_{j \in hp(i) \setminus M(i)} \left\lceil \frac{J_j + \Delta t}{P_j} \right\rceil \cdot C_j \tag{10}$$

where $M(i)$ is the set of tasks in another mode as τ_i that is made mutually exclusive to τ_i using a lock. Using the prove given in Section 7.1 the set of interfering tasks is reduced to tasks in the same mode by using locks. For RR and TDM schedulers a similar expression can be derived where the interference is a summation over the tasks assigned to the same processor. The following equation for the interference for the different types of schedulers is obtained:

$$I_{RR}(i, \Delta t) = \sum_{j \in \mathcal{T}(i) \setminus M(i)} C_j \tag{11}$$

$$I_{TDM}(i, \Delta t) = \left\lceil \frac{C_i}{B_i} \right\rceil \cdot (Q_m - B_i), \quad \text{where } Q_m = B_i + \sum_{j \in \mathcal{T}(i) \setminus M(i)} B_j \tag{12}$$

where B_i is the budget of τ_i within replenishment interval Q_m within mode m and $\mathcal{T}$(i) the set of tasks mapped to the same processor as τ_i excluding τ_i itself. We have shown that

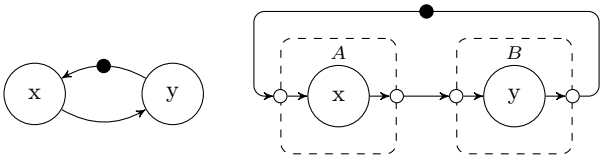
for SPP, RR and TDM schedulers an upper bound on the interference can be derived only consisting of interference of tasks within the same mode as the task being analyzed.

7.1 Mutual exclusive execution using locks

In this section we will prove that the constraints imposed by a lock ensure that tasks in different modes will not interfere with each other and therefore execute mutually exclusive.

For HSDF graphs it has been proven that the number of tokens on a cycle determines the interference of tasks on the cycle [18]. A cycle with one token on it, as shown in Figure 6a, prevents interference between actor x and y and therefore these actors will fire mutually exclusively. In this section it is proven that this also holds for two actors in different blocks in an SVPDF graph, like the one in Figure 6b. If the actors belong to the same block, then they potentially interfere.

The SVPDF graph in Figure 6b has two actors x and y in different modes, A and B, and the edges represent the constraints that are a result of a lock. Any number of modes and tasks within a mode is supported by the lock, but in the proof we will for clarity only consider two modes each containing a single task . In the proof we will make use of the following notation: $a \prec b$ means a dependency from a to b, $s_a(n)$ is the start of firing $n \in \mathbb{N}$ of actor a, $f_a(n)$ is the finish of the n-th firing of actor a. Function $\varphi_A(n) \in \mathbb{N}$ returns the block iteration counter in which firing n of an actor in block A takes place. A new block iteration is started when the next token is consumed by all port actors of the block. Similarly the function $\varphi_B(n) \in \mathbb{N}$ returns the block iteration counter in which firing n of an actor in block B takes place. Also functions that indicate the start and finish time of an iteration of a block are defined.



(a) HSDF graph consisting of actor x and y

(b) SVPDF graph consisting of two modes, A and B, each containing one actor, x or y

Figure 6: The constraints resulting of a lock for an HSDF model (a) and an SVPDF model (b)

By definition an actor x can cause interference on an actor y when actor x finishes its n-th firing later than the start of the m-th firing of y. Similarly an actor y can cause interference on an actor x when actor y finishes its m-th firing later than the start of the n-th firing of x. Two actors x and y fire mutually exclusive if x cannot cause interference on actor y, and y can not cause interference on actor x. That this holds for the actor x and y in the different blocks in Figure 6b can be seen as follows:

The edge e_{xy} represents a dependency. As a result of this edge it holds that:

$$f_x(n) \prec f_A\big(\varphi_A(n)\big) \prec s_B\big(\varphi_B(m)\big) \prec s_y(m), \{n, m \in \mathbb{N} \mid \varphi_A(n) = \varphi_B(m)\} \quad (13)$$

Since it holds that $f_A(\varphi_A(n) - 1) \prec f_A(\varphi_A(n))$ it follows from (13) that:

$$f_x(n) \prec s_y(m), \{n, m \in \mathbb{N} \mid \varphi_A(n) \leq \varphi_B(m)\} \quad (14)$$

Now we have that x can only cause interference on actor y according to (14) if:

$$\varphi_A(n) > \varphi_B(m) \quad (15)$$

A similar reasoning holds for the edge e_{yx} which has one initial token. This initial token causes that the start of the z-th block iteration of A depends on the finish of the $(z-1)$-th block iteration of B. Therefore it holds that:

$$f_y(m) \prec f_B\big(\varphi_B(m)\big) \prec s_A\big(\varphi_A(n) - 1\big) \prec s_x(n), \{n, m \in \mathbb{N} \mid \varphi_B(m) = \varphi_A(n) - 1\} \quad (16)$$

Since $f_B(\varphi_B(m) - 1) \prec f_B(\varphi_B(m))$ it follows that:

$$f_y(m) \prec s_x(n), \{n, m \in \mathbb{N} \mid \varphi_B(m) \leq \varphi_A(n) - 1\} \quad (17)$$

We have that the m-th firing of actor y can only cause interference on the n-th firing of x if the m-th firing of y can finish later than the n-th start of x, which is true according to (17) if:

$$\varphi_B(m) > \varphi_A(n) - 1 \quad (18)$$

Therefore there is no interference from actor x on y and from actor y on x if (15) and (18) hold, thus:

$$\varphi_A(n) - 1 < \varphi_B(m) < \varphi_A(n) \quad (19)$$

Because there is no block iteration counter value $\varphi_B(m)$ for which (19) holds we conclude that the actors x and y do not interfere and therefore fire mutually exclusive, which concludes the proof.

The proof remains valid for q consecutive executions of an actor x starting at firing n. This is because these consecutive executions must belong to the same block iterations such that $\varphi_A(n + q - 1) = \varphi_A(n)$. As a consequence, (19) still holds since there is no block iteration counter value $\varphi_B(m)$ for which actors x and y can interfere.

8. RESPONSE TIMES LARGER THAN PERIOD

The approach presented in the previous sections is only applicable if all tasks have a WCRT smaller than the period of the source. In this section we present conditions under which WCRTs larger than the period are possible, however our impression is that these conditions will be rarely satisfied in practice.

The approach presented in the previous sections does not allow WCRT larger than the period of the source as a result of the barriers. These barriers were introduced to take care that the arrival times of the input data of a block could not affect the response times of the tasks in that block. However, the jitter in the arrival times of the input data can only have an effect on other tasks if these tasks have a sufficiently high priority or have a dependency towards other tasks. If this is not the case then there is no need for the barrier and WCRT larger than the period of the source can be allowed. The simplified WCRT equation derived in Section 7 is invalid for tasks with a response times larger than the period, since multiple executions of a task need to be considered as is accounted for in the equation stated in (2).

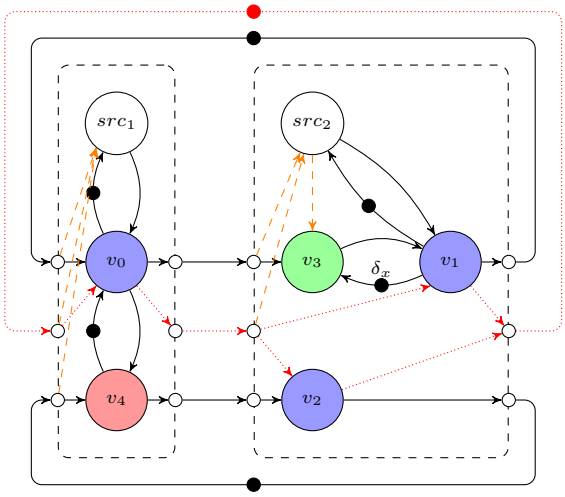


Figure 7: SVPDF model of an application where the WCRT of v_2 can be greater than the period of the source

In Figure 7, a constructed example is shown that illustrates a case in which the barrier can be removed for some inputs of a block. This allows a WCRT larger than the period of the source, which must be compensated for in a subsequent mode. In this example there are five tasks divided across two modes. Task τ_0, τ_1 and τ_2 share a processor, as indicated by the colors of their actors, where τ_0 has a high priority and τ_2 a low priority. The WCET of τ_0, τ_2, and τ_4 is $\frac{1}{4}P$ and τ_1, and τ_3 have a WCET of $\frac{1}{2}P$. Only τ_3 has a Best-Case Execution Time (BCET) not equal to its WCET and is assumed to be 0 and therefore v_3 will introduce a jitter in the enabling time of v_1. Based on this jitter, τ_1 can pre-empt τ_2 not only once, but twice during τ_2's execution, increasing $\hat{R}_2$ to $1\frac{1}{4}P$.

When the dependencies resulting from the barrier would be introduced from the port actor producing the input of v_2 then an $\hat{R}_2 > P$ will lead to the conclusion of infeasibility given that the source period is P. However, in this example τ_2 is allowed to be enabled earlier or later than the source actor starts in its corresponding block. The enabling jitter that can be caused by removing the dependency from the input that is a result of the barrier will not cause interference on other tasks in this example, since τ_2 cannot interfere because it has the lowest priority. Moreover, the jitter of τ_2 is not propagated to the other tasks because there are no dependencies to other tasks and therefore there is no increase of the jitter of other tasks. The large WCRT of τ_2 in this case is compensated by the sufficiently small WCRT of τ_0 and τ_4 in the other mode, because the sum of their WCRTs is within the constraint of one execution of a source in each mode, which takes $2P$.

9. CASE-STUDY

In this section the analysis approach that has been introduced in this paper is demonstrated using a simplified WLAN 802.11p receiver application. The organization of this section is as follows. First we describe the application of which a dataflow model will be derived that includes the constraints that are a result of locks and barriers. We apply temporal analysis to this model to determine the WCRTs of tasks and the buffer sizes. A simulator is used to verify

```
source ADC @ 250 kHz;

loop{
  loop{
    h = detectHeader(ADC);
    vh = validHeader(h);
    NSym' = decodeHeader(h);
  } while(!vh);
  n = 0;
  loop{
    x = fft(ADC);
    y = demap(x);
    z = deint(y);
    w = convDecode(z);
    crc(w);
    n' = n + 1;
  } while(n < NSym);
} while(1);
```

Figure 8: WLAN receiver application

the analysis results and to generate a trace of the executions of the tasks on shared resources. Finally, we show that a pipelined execution of tasks is supported although additional constraints are introduced in the application.

The OIL program of the WLAN application is shown in Figure 8. The application receives its input from a periodic Analog-to-Digital Converter (ADC) running at 250 kHz (a period of 4 μs). This source imposes a throughput constraint on the execution of the application.

The WLAN application contains two modes. Each mode corresponds to a potentially endlessly repeated while-loop.

In the first mode the `detectHeader` function reads symbols from the source until it detects a header of a packet. The `decodeHeader` function extracts the size of the payload from the header while the `validHeader` function determines in parallel if the checksum of the header is valid. Only when a valid header is found the first mode is left as is specified in the loop condition.

The `fft` function in the second modes reads a number of symbols from the source based on the size of the payload and performs a transformation to the frequency domain on each of these symbols. The other functions in this mode perform demapping, deinterleaving, convolution decoding and verification of the CRC. The number of loop iterations in this mode is dependent on the result of the first mode and can be determined before tasks in the second mode start.

The multiprocessor compiler Omphale [3] is used to transform the program shown in Figure 8 into a task graph where each function results in a task. The variables used in the program are converted into buffers to allow a pipelined execution of the tasks. The WCETs of the *detectHeader*, *validHeader* and *decodeHeader* tasks are assumed to be 2 μs, 1 μs and 1 μs as is shown in Table 1a. The *fft, demap, deint, conv* and *crc* tasks in the second mode have an execution time of 3 μs, 1.5 μs, 2 μs, 1.5 μs and 2 μs respectively.

In this case-study we assume there are more tasks than processors. Therefore, multiple tasks are scheduled onto a processor using a scheduling policy. Multiple scheduling policies are used for different processors to demonstrate that the approach presented in this paper can also be used for other schedulers than the SPP scheduler. Tasks *validHeader*, *decodeHeader*, and *fft* are scheduled by an SPP scheduler on one processor. On that processor task *decodeHeader* has the highest priority and task *validHeader* the lowest priority. Another processor runs the tasks *deint*, and *crc* using a RR

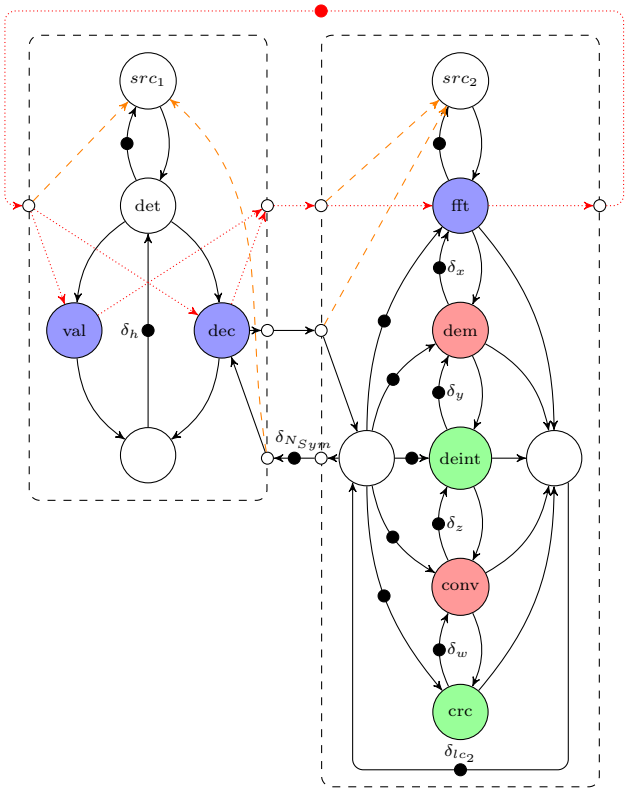


Figure 9: SVPDF model of the application in Figure 8 including additional constraints of a lock and barriers

scheduler. A budget scheduler is used on another processor to schedule the tasks *demap* and *convDecode*. Both tasks are allocated a budget of 0.5 μs every 1 μs. The remaining task *detectHeader* runs on a dedicated processor.

In order to verify the temporal constraint imposed by the periodic source, the compiler also generates an SVPDF model besides the task graph. An actor is introduced for each task in this model as is shown in Figure 9. The blocks in the model correspond to while-loops in the application. The colors of the actors correspond to the mapping to a processor. Buffers are represented as a forward and backward edge containing the number of tokens corresponding to the capacity of the buffer. Since the the *validHeader*, *decodeHeader* and *fft* tasks run on the same processor, but do no not belong to the same mode a lock is added. The analysis flow requires a strictly periodic execution of the source to be able to determine response times for an SPP scheduler. Therefore, additional constraints on the start times of source actors are added in step 2c of the analysis flow. The precedence constraints that results for the lock are represented in the model by the red dotted edges. The additional constraints that are a result of the barriers are shown as the orange dashed edges.

Using the analysis flow as defined in Section 4 and the response time equation (9) of Section 7 we derive WCRTs for the tasks in these two modes. The resulting WCRTs are listed in Table 1a. Table 1b shows the computed buffer capacities given these WCRTs. The last two buffers in the table, δ_{lc1} and δ_{lcc}, represent a buffer in which a loop condition is stored that is read by all tasks in the corresponding mode. The size of these buffers affects the maximum amount of tasks that can execute in parallel.

task	WCET (μs)	$\hat{R}$ (μs)
detectHeader	2.0	2.0
decodeHeader	1.0	1.0
validHeader	1.0	2.0
fft	2.5	2.5
demap	1.5	3.0
deint	2.0	2.0
convDecode	1.5	3.0
crc	2.0	4.0

(a) Task execution times and derived response times

buffer	capacity
δ_h	1
δ_{vh}	1
δ_x	2
δ_y	2
δ_z	2
δ_w	2
$\delta_{N_{Sym}}$	1
δ_{lc_1}	1
δ_{lc_2}	4

(b) Derived buffer capacities

Table 1: Temporal analysis results of the WLAN application

We use the high-level system simulator HAPI to verify the obtained analysis results. HAPI was initially a dataflow simulator [1], but was recently extended with the addition of processor sharing, which allows for the simulation of task graph executions on arbitrary platforms. No constraint violations were detected for the derived buffer capacities because the source could fire strictly periodically. The traces generated with the HAPI simulator are shown in Figure 10. In this figure there is one trace for each processor in which the currently running task and the iteration number of the task is shown. In each trace pre-emptions based on either priorities or depletion of budget are indicated by X's. The trace in Figure 10 shows pipeline parallelism for example at a time of 24 μs, at which the *fft* task already processes the next symbol of the source whereas the CRC of a previous symbol is being computed by the *crc* task.

Pipelining of the WLAN application would be impossible for certain assignments of tasks to processors. The additional constraints in the application as a result of barriers and locks can prevent pipeline parallelism inside a mode. In the example the first task in the second mode (*fft*) is assigned to the same processor as tasks in the first mode. The combined constraints of the buffer from the source to *fft* and the lock and barrier, represented by the red dotted and orange dashed edge in Figure 9, require *fft* to process a sample from the source within one period when switching to the first mode. When for example task *crc* would be mapped to that processor instead of task *fft*, multiple tasks need to be finished within the same period. In the dataflow model, the path without initial tokens from actor src_2 via *fft*, *demap*, *deint* and *convDecode* to actor *crc* would than be a bottleneck and limit pipelining since task *crc* would then release the lock. The barrier in the first mode requires the lock to be acquired before the source in the same mode finished its first execution. Therefore, it can be concluded that the amount of achievable pipeline parallelism depends on the mapping of tasks to processors when an SPP scheduler is used.

10. CONCLUSION

This paper presents a compositional temporal dataflow analysis approach for cyclic stream processing applications with modes executed on multiprocessor systems that use SPP scheduling.

The analysis approach is based on the ability to independently characterize the temporal behavior of modes. This is

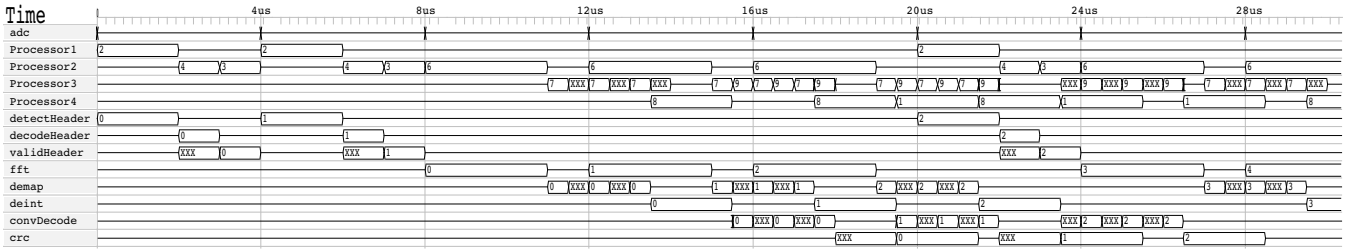


Figure 10: Execution trace of the WLAN application from Figure 8 including a lock and barriers

ensured by adding locks in the application which make the execution of tasks that belong to different modes but executed on the same processor mutually exclusive. Furthermore, barriers are added such that together with the locks the interference of the tasks in a mode is independent of tasks belonging to other modes. The additional constraints introduced by the barriers and locks guarantee that composition of modes does not change their individual characterization. As a result, applications containing a hierarchy of modes can be described in an SVPDF model. This model can be analyzed by recursively applying existing dataflow analysis techniques, to determine the worst-case temporal behavior. The SVPDF model and the parallel implementation including locks and barriers are generated by a multiprocessor compiler.

Furthermore it is shown that the approach allows for response times of tasks larger than the period of the source. It is also shown that systems in which a combination of budget schedulers and SPP schedulers are applied can be analyzed.

The applicability of the approach is demonstrated using a WLAN 802.11p receiver application. We show that this application can be executed pipelined despite the inserted locks and barriers. The analysis results are verified using a dataflow simulator.

11. REFERENCES

[1] M. J. G. Bekooij, S. Parmar, and J. L. van Meerbergen. Performance guarantees by simulation of process networks. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 10–19. ACM, 2005.

[2] M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *IEEE/ACM/IFIP Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2010.

[3] S. J. Geuns, J. P. H. M. Hausmans, and M. J. G. Bekooij. Automatic dataflow model extraction from modal real-time stream processing applications. In *Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 2013.

[4] S. J. Geuns, J. P. H. M. Hausmans, and M. J. G. Bekooij. Temporal analysis model extraction for optimizing modal multi-rate stream processing applications. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2014.

[5] Q. Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Systems*, 41(3):181–194, 2009.

[6] J. P. H. M. Hausmans, S. J. Geuns, M. H. Wiggers, and M. J. G. Bekooij. Temporal analysis flow based on an enabling rate characterization for multi-rate applications executed on MPSoCs with non-starvation-free schedulers. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 108–117. ACM, 2014.

[7] J. P. H. M. Hausmans, M. H. Wiggers, S. J. Geuns, and M. J. G. Bekooij. Dataflow analysis for multiprocessor systems with non-starvation-free schedulers. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*. ACM, 2013.

[8] G. Kuiper, S. J. Geuns, and M. J. G. Bekooij. Utilization improvement by enforcing mutual exclusive task execution in modal stream processing applications. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*. ACM, 2015.

[9] P. S. Kurtin, J. P. H. M. Hausmans, and M. J. G. Bekooij. Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications. In *Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 2016. To appear.

[10] A. Lele, O. Moreira, and K. van Berkel. FP-scheduling for mode-controlled dataflow: a case study. In *Design, Automation and Test in Europe (DATE)*, pages 1257–1260. EDA Consortium, 2015.

[11] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-time systems*, 2004.

[12] L. Sha et al. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264, 1989.

[13] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptively scheduled systems. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 100–109. IEEE, 1992.

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

[15] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 11–22. ACM, 2007.

[16] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Monotonicity and run-time scheduling. In *ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2009.

[17] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Buffer capacity computation for throughput-constrained modal task graphs. *ACM Trans. on Embedded Computing Systems (TECS)*, 10(2):17, 2010.

[18] P. S. Wilmanns, J. P. H. M. Hausmans, S. J. Geuns, and M. J. G. Bekooij. Accuracy improvement of dataflow analysis for cyclic stream processing applications scheduled by static priority preemptive schedulers. In *Euromicro Conf. on Digital System Design Architectures, Methods and Tools (DSD)*. IEEE, 2014.