

# Dynamic Response Time Optimization for SDF Graphs

Dirk Ziegenbein, Jan Uerpmann, Rolf Ernst\*  
TU Braunschweig  
{ziegenbein, uerpmann, ernst}@ida.ing.tu-bs.de

## Abstract

*Synchronous Data Flow (SDF) is a well-known model of computation that is widely used in the control engineering and digital signal processing domains. Existing scheduling methods are mainly static approaches that assume full knowledge of the environment, e. g. data arrival times. In a growing number of practical cases like internet multimedia applications there exists only partial knowledge of the environment, e. g. average data rates. Here, only dynamic scheduling can yield optimal results. In this paper, we propose a new dynamic scheduling method that minimizes the maximal response time of the system. It is a generalization of a deadline revision method to allow treatment of data-dependent tasks using EDF scheduling. The applicability and benefit of the new approach is shown using a real-world example.*

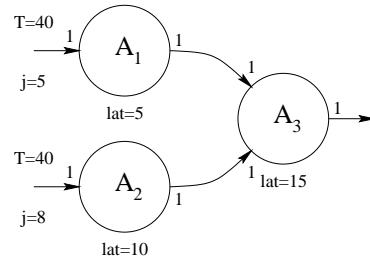
## 1 Introduction

Synchronous Data Flow (SDF)[6] is a well-known model of computation that is widely used in the control engineering and digital signal processing domains. The representation of an application as a set of actors (or processes) communicating via buffers expresses explicitly the degree of parallelism in the system. There is a very sound understanding of the theory and a lot of methods for analysis and scheduling have been developed. This also shows in a wide variety of design tools that are based on SDF, e. g. COSSAP by Synopsys, SPW by Cadence Alta or DSP DesignStation by Mentor Graphics.

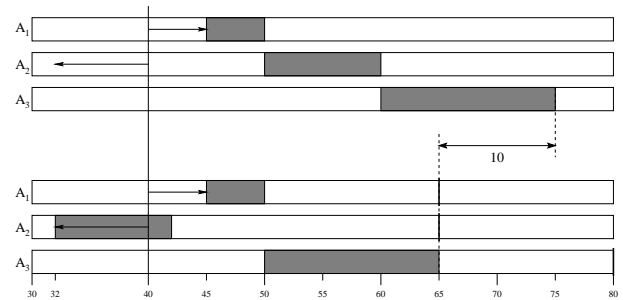
SDF graphs allow an arbitrary order of actor (process) executions only subject to the availability of input data. This property can be exploited for throughput and/or buffer size optimization. Assuming infinite input streams (e. g. stored in a buffer) or input with exactly predictable timing, such as periodic input, optimality can even be reached with static scheduling, i. e. a static order of process executions. Thus, the static implementation methods assume either large, costly input buffers or a complete knowledge of the environment with exact data arrival times.

In many important embedded applications, input data arrive in a non-periodic sequence. Prominent examples are burst modes and packet based transmission, such as in automotive

(CAN bus) or internet-based applications. In both cases, the transport medium lacks real-time characteristics to guarantee exact timing. Under these circumstances, a static order of process executions cannot provide optimal solutions.



**Figure 1: Example SDF graph with uncertain environment information**



**Figure 2: Gantt charts for SDF graph of Fig. 1 for static (top) and dynamic (bottom) scheduling**

Fig. 1 shows a very simple example with three actors to demonstrate the possible shortcomings of static scheduling of SDF graphs when the environment information is uncertain. The input tokens arrive at both inputs every 40 time units on the average but have a jitter of 5 and 8 time units, respectively. For a static schedule, the execution order of actors  $A_1$  and  $A_2$  is fixed off-line. Thus, an unfavorable situation (input for  $A_1$  is late and for  $A_2$  early) as depicted in the top Gantt chart of Fig. 2 can arise that the system waits on an input token for  $A_1$  although  $A_2$  is ready to execute. Using dynamic scheduling, the execution order of the actors is only partially fixed according to the data dependencies and the early arrival of the input token for  $A_2$  can be used to minimize the response time (see lower Gantt chart in Fig. 2).

As the example shows, dynamic scheduling of SDF graphs is preferable for uncertain environments. Here and in the following, we assume that the actors are sufficiently large pro-

\*This paper was published at ICCAD 2000.

cesses, otherwise the advantage will be spoiled by context switch overhead.

Other than static process ordering, dynamic scheduling algorithms are very sensitive to the optimization goals. It is easy to obtain maximum throughput for SDF graphs by simply executing any activated process with sufficient input data. In this case, we utilize the SDF property of an arbitrary process sequence. If more than one process is activated, which is typical for SDF systems, then we have a choice which process to execute. This choice can be used to optimize for a second optimization goal, i. e. buffer size minimization or response time optimization. In this work, we focus on optimizing the guaranteed system response time. Applications are (interactive) multimedia systems and any kind of control engineering application which requires a strict upper bound on their latencies or response times, such as in automotive systems.

There is a lot of expertise in the field of dynamic scheduling to guarantee deadlines or minimize response times, but the algorithms are not applicable to SDF graphs since they assume acyclic single-rate process systems. To the best of our knowledge, there are no dynamic scheduling methodologies that support multi-rate and cyclic data dependencies, two key features of SDF graphs. Thus, we propose a new method for dynamic scheduling of SDF graphs.

Our approach is a generalization of a method by Blazewicz [2] who showed how to revise deadlines of task with general precedence constraints such that earliest deadline first (EDF) scheduling yields minimal response times. While the original method only supports precedence constraints between tasks with the same execution rate, we extend the method to cover multi-rate data dependencies and cycles in the graph structure. The main advantage of our approach is to reduce the worst case system response time in comparison with existing static approaches.

The rest of the paper is organized as follows. After an overview of related work, the problem to be solved is defined in Section 3. In Section 4, Blazewicz's method as well as our extensions are presented and explained using simple examples. Our approach is demonstrated using a real-world autonomous vehicle controller in Section 5. The paper ends with a conclusion about the applicability of our approach and an outlook on further work.

## 2 Related Work

Synchronous Data Flow (SDF)[6] is a well-known model of computation that is widely used in the control engineering and digital signal processing domains. There is a very sound understanding of the theory and a lot of methods for analysis and scheduling have been developed. Lee and Messerschmitt [5] present methods for checking for the existence of a bounded memory schedule and propose algorithms for scheduling SDF graphs on uni- and multi-processors. Ha and Lee [4] extend these algorithms for actors with data-dependent execution times. Furthermore, there are several extensions of the SDF model of computation with respect to data-dependent data movement, e. g. Boolean Data Flow [3].

In contrast to SDF related methods that are mostly performed at compile-time and require full knowledge of the environment, dynamic scheduling methods are performed at run-time and are widely used in the context of real-time operating systems. A very good overview of standard dynamic scheduling methods is given by Stankovic et. al. in [9]. While the standard approaches like EDF and RMS are mostly restricted to independent tasks, extensions to handle data dependencies have been made by Blazewicz [2] for EDF and by Yen and Wolf [11] for static priority preemptive methods. The latter approach has been extended by Pop, Eles and Peng [8] to cover control dependencies.

## 3 Problem Formulation

The system to be scheduled is given as an SDF graph[6]. This is a directed graph with nodes, called actors, representing computations and edges representing data communication via unbounded FIFO buffers that may have a number of preassigned tokens  $\hat{d}$  called delays. An actor  $A$  has an associated firing rule that allows execution only when there is sufficient input data available on its incoming edges. At each execution it consumes (blocking read) and produces (non-blocking write) a fixed number of data tokens,  $r$  and  $s$  respectively, on their connected edges. Each actor  $A$  has a worst-case execution latency  $\text{lat}_A$ . The graph also has external inputs and outputs through which data tokens are received from the environment and sent to the environment, respectively.

Besides the system, the description of the problem also has to include the environment. In the context of this paper, we assume that each input of the SDF graph has an associated input data rate. An input data rate is described by two parameters:

- **the mean period  $\bar{T}$**   
The mean period is the average time between the arrival of two consecutive data tokens at the corresponding input of the system.
- **the jitter  $j$**   
The jitter defines the maximum time difference between the actual arrival of a data token and its arrival according to the mean period. If the jitter intervals overlap we assume that the data tokens arrive in the correct order.

For example, at an input with an associated input data rate characterized by  $\bar{T} = 40$  and  $j = 5$  tokens arrive every 40 time units on the average but may be 5 time units "late" or "early". Both parameters represent the average behavior of the environment and the bounds in which this behavior may vary. Note that the jitter has no direct influence on the proposed scheduling method but on the achieved gain compared to existing static methods, since zero jitter stands for complete knowledge of the environment for which static methods yield optimal results as well.

Well-formed SDF graphs and a "correct" environment are assumed, i. e. the ratio of two input mean periods has to be equal to the ratio of the execution rates of the actors that are

connected to the corresponding inputs. Otherwise, the system is ill-formed and the input buffers are not bounded.

The proposed scheduling method maps the SDF graph onto a uniprocessor with the goal to minimize the guaranteed overall response time of the system.

## 4 Response Time Optimization

In this section, the proposed approach is presented and explained using simple examples. After an introduction of the scheduling method by Blazewicz on which the approach is based, the extensions to cover multi-rate and cyclic data dependencies are presented.

### 4.1 Blazewicz's Algorithm

A widely used preemptive scheduling algorithm is the earliest deadline first (EDF) scheduling policy that assigns the highest priority to the task with the minimum remaining time until its deadline. Liu and Layland [7] showed that EDF scheduling is optimal and always achieves full processor utilization for independent tasks.

Since an essential feature of SDF graphs is the presence of data dependencies between actors the simple EDF policy is not applicable for our goal. However, Blazewicz [2] proposed an optimal method how to adjust deadlines and start times for dependent tasks such that precedence constraints are encoded in the revised deadlines. Then, these tasks can be scheduled using the simple EDF policy such that the overall system response time is minimized. According to Blazewicz, the deadlines  $d_i$  and start times  $a_i$  have to be revised as follows:

$$d_i^* = \min\{d_i, \min(d_j^* - \text{lat}_{A_j}; A_i \rightarrow A_j)\}$$

*starting from tasks with no successor and processing step by step all tasks with their successors already been processed*

$$a_i^* = \max\{a_i, \max(a_j^* + \text{lat}_{A_j}; A_j \rightarrow A_i)\}$$

*starting from tasks with no predecessor and processing step by step all tasks with their predecessors already been processed.*

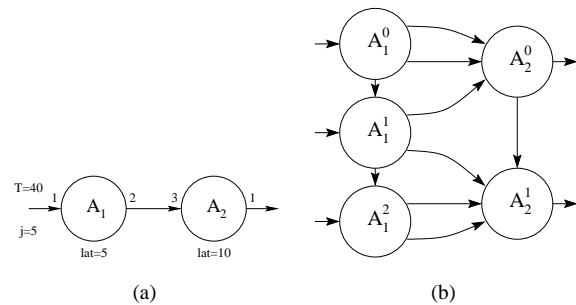
The revised start times ensure that no task will be executed too early and violate any precedence constraints, e. g. preventing a task from reading a value out of shared memory before it is updated. However, start times do not have to be considered for the scheduling of SDF graphs, since the precedence constraints in SDF graphs are data dependencies with blocking reads, i. e. no actor is able to execute before all its input data is present (as required by the firing rules) and thus all its data dependencies are observed. Thus, in the following only Blazewicz's method for revision of deadlines is extended although the ideas are valid for the adaption of start times, too.

Since our goal is to minimize the guaranteed overall system response time, there are no explicit task deadlines specified. Thus at first sight, EDF which assigns priorities only based on deadlines seems not to be applicable at all. But not the absolute value of its deadline is important for the priority

assignment of a task but the relative value in comparison with the deadlines of the other tasks. Since relative deadlines are obtained by performing Blazewicz's method, EDF is applicable for our problem. In order to obtain meaningful values for the deadlines, we initialize the deadlines of all actors connected to outputs to be the duration of one mean macro period. The duration of a mean macro period  $\bar{T}_{\text{macro}}$  can be easily determined by multiplying the mean period for one input with the number of executions per macro period  $q$  of the actor connected to this input ( $\bar{T}_{\text{macro}} = \bar{T}_i \cdot q_i$ ). The determination of  $q$  is shown in section 4.2.

Using the example system depicted in Fig. 1, Blazewicz's deadline revision is explained in the following. The deadline of  $A_3$  is initialized to  $d_3 = \bar{T}_{\text{macro}} = 40$ . Then,  $A_2$ 's deadline is determined as  $d_2 = \min\{\infty, (40 - 15)\} = 25$ . Since the dependencies of  $A_1$  and  $A_2$  with respect to  $A_3$  are the same,  $d_1$  evaluates to 25 as well. Thus, actors  $A_1$  and  $A_2$  are on the same priority level, i. e. the order of execution depends on the order of their respective incoming data tokens. In the example situation depicted in the bottom Gantt chart in Fig. 2,  $A_2$ 's input data arrives before  $A_1$ 's. This results for the EDF schedule in the execution order  $A_2A_1A_3$ .

Blazewicz's algorithm is targeted for task graphs with general precedence constraints. However, multi-rate data dependencies as they are common in SDF graphs are not covered. Furthermore, the formulation of the deadline revision procedure does not allow cycles, also a common feature of SDF graphs. In the following subsections, extensions of Blazewicz's algorithm are proposed to cover multi-rate dependencies as well as cycles.



**Figure 3: Example for a graph with a multi-rate data dependency**

### 4.2 Multi-Rate Extension

The basic idea of the extension of Blazewicz's method to cover multi-rate data dependencies is that for each multi-rate graph there exists an equivalent single-rate graph that can be constructed by unfolding the graph [1] for the duration of a macro period. A macro period is the time in which the SDF graph after a series of actor executions again reaches its initial state. The constructed single-rate graph has a node for each instance of an actor in the macro period and single-rate data dependencies between them. Since there are only single-rate dependencies,

performing Blazewicz's method on this unfolded graph yields optimal deadlines for each actor instance. Thus, optimality can still be guaranteed for graphs with multi-rate data dependencies.

For the example system of Fig. 3a), a macro period consists of three executions of actor  $A_1$  and two executions of actor  $A_2$ . The corresponding unfolded single-rate graph is depicted in Fig. 3b). Note that the multi-rate data dependency has been transformed into six single-rate dependencies. Additionally, precedence constraints between different instances of the same actor have been introduced which ensure that the  $k$ th instance of an actor is executed before its  $(k + 1)$ th instance.

To avoid the possibly computationally intensive explicit construction of this unfolded graph, we extend Blazewicz's deadline revision formula to cover all actor instances in a macro period. This can be divided into two subproblems:

- **Find number of instances of each actor per macro period**

This was solved by Lee and Messerschmitt in [5]. They create a topology matrix  $\Gamma$  that has a column for each actor and a row for each edge in the graph. The  $(j, i)$ th entry in the matrix denotes the number of data tokens produced by actor  $A_i$  on edge  $e_j$  per execution. If actor  $A_i$  consumes tokens from edge  $e_j$ , the entry is negative, and if  $A_i$  is not connected to  $e_j$ , the entry is zero. By solving  $\Gamma \vec{q} = \vec{0}$  for execution number vector  $\vec{q}$  with  $\vec{0}$  being a vector full of zeros, the number of instances  $q_i \in \{1, 2, 3, \dots\}$  for each actor  $A_i$  per macro period is obtained. In the following formulas, we use for each actor  $A_i$  the maximum instance index in a macro period  $k_{max,i} \in \{0, 1, 2, \dots\}$  which is computed by  $q_i - 1$ . For the example in Fig. 3, this evaluates to  $k_{max,1} = q_1 - 1 = 3 - 1 = 2$  and  $k_{max,2} = 2 - 1 = 1$ .

- **Find dependent instances for each multi-rate data dependency**

A single-rate data dependency from actor  $A_i$  to actor  $A_j$  directly defines that the  $k$ th instance of  $A_j$  has to be preceded by the  $k$ th instance of  $A_i$ .  $\hat{d}$  delays on the edge representing this data dependency result in a dependency between the  $(k + \hat{d})$ th instance of  $A_j$  and the  $k$ th instance of  $A_i$  (in the following denoted as  $A_i^k$ ). However, for a multi-rate data dependency from actor  $A_i$  to actor  $A_j$ , several instances of actor  $A_j$  may depend on the same instance of  $A_i$  or one instance of  $A_j$  may depend on several instances of actor  $A_i$ . For the determination of these dependencies, the input and output data rates  $s_{i,j}$  and  $r_{i,j}$  as well as the number of delays  $\hat{d}_{i,j}$  on the edge connecting the actors have to be considered.

For the derivation of the deadline for the  $k_i$ th instance of actor  $A_i$ , the first instance of actor  $A_j$  that is dependent on the execution of  $A_i^{k_i}$  has to be determined. Thus, it has to hold that  $A_i^{k_i}$  produces the last token needed for the execution of instance  $A_j^{k_j}$ , i. e. the correct instance index is the smallest value of  $k_j \in \mathbb{N}_0$  that satisfies the following

expression

$$k_i \cdot s_{ij} + \hat{d}_{ij} < (k_j + 1) r_{ij}$$

where the left side denotes the number of tokens produced by  $A_i$  before the  $k_i$ th instance, including delays, and the right side denotes the number of tokens read by  $A_j$  after the  $k_j$ th instance. Solving for  $k_j$  leads to

$$k_j > \frac{k_i \cdot s_{ij} + \hat{d}_{ij}}{r_{ij}} - 1$$

The smallest  $k_j$  which meets this inequation is

$$k_j = \left\lceil \frac{k_i \cdot s_{ij} + \hat{d}_{ij}}{r_{ij}} \right\rceil$$

Thus, the deadline  $d_i^{k_i}$  of the  $k_i$ th instance of actor  $A_i$  depends on the  $\left\lceil \frac{k_i \cdot s_{ij} + \hat{d}_{ij}}{r_{ij}} \right\rceil$ th instance of actor  $A_j$ .

Delays on a data dependency may lead to a dependency on an instance that is part of a higher macro period. Since deadlines are only determined for instances in one macro period, the deadline of a higher macro period instance evaluates to

$$d_j^k = d_j^{k-q_j} + \left\lceil \frac{k}{q_j} \right\rceil \cdot \bar{T}_{macro} \quad \text{for } k \geq q_j$$

Additionally, the precedence constraints between instances of the same actor have to be considered. This results in the following expression

$$d_i^k \leq d_i^{k+1} - \text{lat}_{A_i}$$

that follows Blazewicz's method for deadline revision and ensures that the  $k$ th instance is executed before the  $(k + 1)$ th instance of actor  $A$ .

Altogether, this results in the modified formula for deadline revision that is depicted in Fig. 4 and has to be computed for all actor instances  $A_i^k$  in one macro period starting from actors with no successors and from  $k = k_{max,i}$  down to  $k = 0$ . This formula can be seen as a generalization of Blazewicz's formula since it yields the same results for single-rate precedence constraints and treats multi-rate constraints just as their corresponding single-rate constraints.

For the example in Fig. 3, the deadline of the last instance of  $A_2$  is initialized to  $d_2^1 = \bar{T}_1 \cdot q_1 = 40 \cdot 3 = 120$ . According to the modified formula the other deadline  $d_2^0$  is set to  $d_2^0 = d_2^1 - \text{lat}_{A_2} = 110$ . Having evaluated the deadlines of the last actor  $A_2$  now the deadlines of  $A_1$  have to be calculated. To obtain the deadline  $d_1^2$ , one has to determine the corresponding  $d_2^{k_j}$  which is  $k_j = \left\lfloor \frac{2 \cdot 2 + 0}{3} \right\rfloor = 1$ . Thus  $d_1^2$  results in  $d_1^2 = \min \{\infty, d_2^1 - \text{lat}_{A_2}\} = 110$ . Similarly, the other deadlines of  $A_1$  are set to  $d_1^1 = 100$  and  $d_1^0 = 95$ .

$$d_i^k = \min \left\{ d_i^k, d_i^{k+1} - \text{lat}_{A_i}, \min \left\{ d_j^k \left\lfloor \frac{k \cdot s_{ij} + d_{ij}}{r_{ij}} \right\rfloor - \text{lat}_{A_j}; A_i \rightarrow A_j \right\} \right\}$$

starting from actors with no successors and with falling  $k$ .

**Figure 4: Modified deadline revision formula**

### 4.3 Cycle Extension

In its original formulation, Blazewicz's method does not consider cycles of precedence constraints. This is well understandable, since for general precedence constraints a cycle would obviously result in a deadlock. But for precedence constraints originating from SDF data dependencies, cycles are meaningful due to the concept of delays.

As already shown above, delays cause a data dependency to result in a precedence constraint ranging to a later instance of the receiving actor. Delays are already covered by the multi-rate extension above, but cycles can not be treated. This is due to the formulation of Blazewicz's algorithm that requires all deadlines of succeeding actors to be known for the determination of an actor's deadline which is impossible for actors in a cycle. Again, Blazewicz's idea holds also for cyclic data dependencies, the restriction is only based on the formulation of the algorithm.

The solution is to use an algorithm that iteratively propagates and updates the revised deadlines according to the modified deadline revision formula until no changes occur. This algorithm is depicted below:

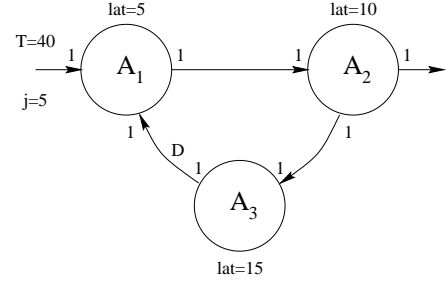
```

Dirty := {A1, ..., AN}
while Dirty ≠ ∅
{
  choose Ai ∈ Dirty
  {
    for all k from kmax,i downto 0
    {
      dTemp := dik
      dik = ... (formula of Fig. 4)
      if dTemp ≠ dik
        then Dirty+ = Predecessor(Ai)
      if latAi > dik then break
      DELAY_FAILURE
    }
  }
}

```

The set `Dirty` contains all processes that still have to be processed. Whenever the deadline of an actor instance is changed, all direct predecessors are added to the `Dirty` set, since the changed deadline might also effect the deadlines of their instances. This is done until the set `Dirty` is empty, i. e. no changes occurred. In case of insufficient delays in a cycle, which is a sign of an ill-formed SDF graph that would deadlock, the algorithm does not terminate, if the break condition  $\text{lat}_i > d_i^k$ , i. e. the deadline is too early to be fulfilled, is not introduced. Together with this break condition, the algorithm

is guaranteed to terminate since the deadlines monotonically decrease due to the `min`-operator.



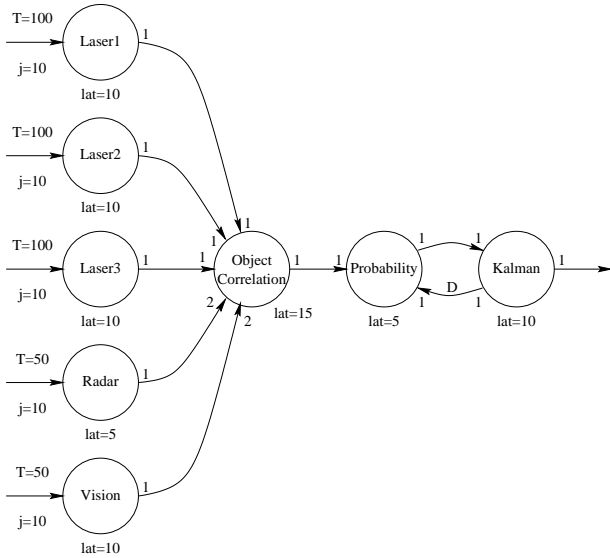
**Figure 5: Example SDF graph with cycle**

The algorithm is demonstrated using the simple example of Fig. 5. The deadline of actor  $A_2$ , being an actor connected to an output, is initialized with  $d_2^0 = \bar{T}_{\text{macro}} = 40$ . Next the deadline of  $A_1$  is calculated as  $d_1^0 = d_2^0 - \text{lat}_{A_2} = 30$ . The deadline  $d_3^0$  corresponds with  $k_j = \lfloor \frac{0 \cdot 1 + 1}{1} \rfloor = 1$  and  $d_1^1 = d_1^{1-1} + \lfloor \frac{1}{1} \rfloor \cdot 40 = 70$  to  $d_3^0 = 70 - 5 = 65$ . After this it has to be checked if the new deadline of  $A_3$  modifies the deadline of  $A_2$ . Because  $d_2^0$  is determined as  $d_2^0 = \min \{40, 65 - 15\} = 40$  it is not changed and the algorithm terminates.

## 5 Example

In this section, we apply our method to a real-life automotive application example, the obstacle recognition part of an autonomous vehicle controller for test tracks [10] shown in Fig. 6. The input data for the object recognition system is provided by five sensors (3 laser scanners, 1 radar sensor, 1 stereo vision sensor) that operate asynchronously and transmit their acquired data via several CAN buses to the system. The actors connected to each input perform the separate recognition of objects. The actor called object correlation decides which objects from different sensors represent in fact the same obstacle. Once the objects are matched, they are combined by actors Probability Estimation and Kalman Filter and then sent to the path planning module.

Fig. 7 shows Gantt charts for both a static schedule with off-line fixed execution order of actors (top) and for a schedule obtained by our proposed scheduling method (bottom). The arrows denote deviations of incoming data from their average arrival time represented as weak vertical lines. The solid vertical lines in the bottom Gantt chart denote the revised deadlines for the respective actors. The Gantt charts show how the static



**Figure 6: Obstacle recognition part of autonomous vehicle controller**

schedule has to wait for the delayed input coming from the radar sensor while the dynamic schedule already executes object recognition for the vision sensor and a laser scanner. This results in a 15% lower system response time.

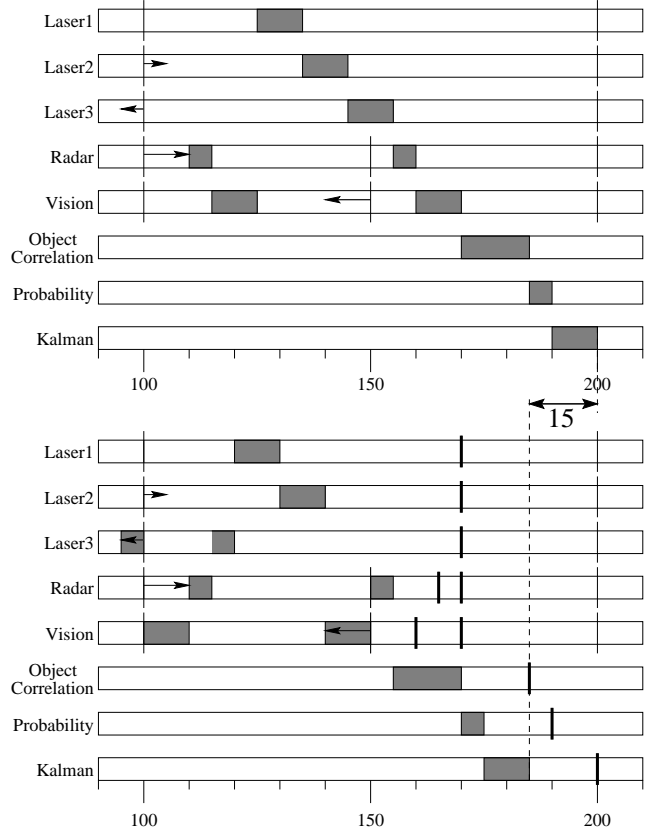
## 6 Conclusion

We proposed a new dynamic scheduling algorithm for SDF systems with uncertain input arrival times that utilizes the Kahn graph properties of SDF to optimize guaranteed system response times while keeping maximum throughput. The approach is based on an existing deadline revision method that enables EDF scheduling for tasks with single-rate data dependencies and generalizes this method for SDF graphs by allowing multi-rate and cyclic data dependencies.

In an automotive application example featuring a CAN field bus as the main source for a jittered signal arrival time, the proposed approach yields 15% lower response times. Highly dynamic environments using internet communication links or buses with burst-modes are likely to show even higher gains.

## References

- [1] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclostatic dataflow. *IEEE Transactions on Signal Processing*, 44(2), February 1996.
- [2] J. Blazewicz. *Modeling and Performance Evaluation of Computer Systems*, chapter Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines. North-Holland, Amsterdam, 1976.
- [3] J. Buck and E.A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *Proceedings of ICASSP'93*, 1993.



**Figure 7: Gantt charts for static (top) and dynamic (bottom) schedules of obstacle recognition example**

- [4] S. Ha and E.A. Lee. Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration. *IEEE Transactions on Computers*, 40(11):1225–1238, November 1991.
- [5] E. A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), January 1987.
- [6] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, volume 75, pages 1235–1245, September 1987.
- [7] C. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of the ACM*, pages 46–61, 1973.
- [8] P. Pop, P. Eles, and Z. Peng. Performance estimation for embedded systems with data and control dependencies. In *Proceedings Eighth International Workshop on Hardware/Software Co-Design (Codes/CASHE '00)*, San Diego, May 2000.
- [9] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6), June 1995.
- [10] H. Weisser, P. Schulenberg, R. Bergholz, and U. Lages. Autonomous driving on vehicle test tracks: Overview, motivation and concept. In *International Conference on Intelligent Vehicles*, volume 2, pages 439–443, 1998.
- [11] T. Yen and W.H. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1125–1136, November 1998.