Context-Aware Performance Analysis for Efficient Embedded System Design

Marek Jersak, Rafik Henia, Rolf Ernst
Technische Universität Braunschweig
Institut für Datentechnik und Kommunikationsnetze (IDA)
D-38106 Braunschweig, Germany
{jersak, henia, ernst}@ida.ing.tu-bs.de

Abstract

Performance analysis has many advantages in theory compared to simulation for the validation of complex embedded systems, but is rarely used in practice. To make analysis more attractive, it is critical to calculate tight analysis bounds. This paper shows that advanced performance analysis techniques taking correlations between successive computation or communication requests as well a correlated load distribution into account can yield much tighter analysis bounds. Cases where such correlations have a large impact on system timing are especially difficult to simulate and, hence, are an ideal target for formal performance analysis.

1. Introduction

Performance validation is key during architecture design and function implementation of state-of-the-art embedded systems. It has to consider the host of non-functional dependencies introduced through processor and bus scheduling. Most existing performance validation approaches rely on simulation and hence suffer from high running times, incomplete coverage and failure to identify corner cases. The problems are aggravated for applications with many scenarios of operation, since combinatorially more cases have to be simulated.

A promising alternative to simulation is formal performance analysis. It can calculate conservative lower and upper bounds for performance values over a range of scenarios and thus guarantees corner-case coverage. Additionally, performance analysis often runs considerably faster than simulation, making it well suited for exploration in early design stages.

If performance analysis seems so attractive in theory compared to simulation, why is it rarely used in practice? A major reason is the fact that performance analysis can be very pessimistic, because it ignores certain correlations between consecutive task activations. Such pessimism sheds a negative light on performance analysis in the embedded systems community, which does not accept over-dimensioned solutions. Therefore, to make performance analysis more attractive, it is critical to calculate tight analysis bounds which are very close to the true performance corner-cases.

A typical performance analysis technique assumes that every activation of a task leads to the worst-case execution time of the task. I.e., it ignores that there may be paths through the control-flow graph of the task leading to shorter execution times. Performance analysis also typically assumes a very pessimistic worst-case load distribution over time. I.e., it ignores that certain task activations often cannot happen at the same time, which evens out the load distribution to a certain extent. We call such correlations *system contexts*.

This paper shows that advanced performance analysis techniques taking system contexts into account can yield much tighter analysis bounds, making analysis an attractive validation option during embedded system design. The designer then automatically profits from the fundamental advantages that analysis has over performance simulation. Performance analysis can also reveal system-level effects due to small local changes, e.g. a slight change in the worst-case execution time of a task.

After a brief overview of the state-of-the-art in performance analysis, two different types of contexts and the analysis improvements that can be obtained are discussed in sections 4 and 5. In section 6 it is shown that additional improvement is possible if contexts can be combined. The design of a hypothetical set-top box (section 3) is used as an example. The paper concludes with a summary and outlook.

2. State-of-the-Art in Performance Analysis

The goal of performance analysis is to verify that an embedded system implementation meets all response-time and throughput constraints, e.g. end-to-end deadlines. Additionally, it can be used to obtain reliable values for worst-case processor and bus loads, required memories etc.

The performance of tasks sharing a single component (e.g. a CPU or a bus) can be analyzed using so-called scheduling analysis techniques. Consider the following equation which gives the worst-case response time r_i of a lower priority task i due to a worst-case number of interrupts by all higher priority task $j \in hp(i)$.

$$r_i = C_i + \sum_{\forall j \in hp(i)} n_j(r_i) \times C_j \qquad (1)$$

$n_j(r_i)$ is the maximum number of activating events for task j arriving during r_i. C_i, C_j are the worst-case core execu-

tion times (WCET), i.e. assuming no interrupts, of tasks i and j. The equation holds only if r_i is smaller than the minimum distance between two activations of task i. For more general cases see e.g. Tindell [9].

Since r_i appears on both sides of the equation, it is usually calculated iteratively, as shown in the following algorithm.

```
1  WorstCaseResponseTime(LowPriorTask){
2    NewResponseTime = LowPriorTask.WCET;
3    Do{
4      OldResponseTime = NewResponseTime;
5      for(int i = 0; i < HighPriorTasksList.size; i++){
6        HighPriorTask = HighPriorTasksList.get(i);
7        MaxActivations = max number of activations
         of HighPriorTask during OldResponseTime;
8        InterruptTime =
         MaxActivations * HighPriorTask.WCET;
9        NewResponseTime =
         NewResponseTime + InterruptTime;
10     }
11   }while(NewResponseTime > OldResponseTime)
12   return NewResponseTime;
13 }
```

Scheduling analysis algorithms are not directly applicable to a heterogeneous multi-component architecture with different scheduling and resource-sharing strategies. Eles et al. have extended existing techniques to allow performance analysis of special heterogeneous architectures, e.g. fixed-priority-scheduled CPUs connected via a TDMA-scheduled bus [5]. Richter [6, 7] takes a more general approach that allows to couple different existing performance analysis techniques for an arbitrarily complex architecture. Chakraborty et al. take a somewhat similar approach using a real-time calculus [2]. Jersak [3] has extended [6, 7] to allow performance analysis for applications with complex task dependencies, including multiple activating inputs and multi-rate data dependencies with intervals.

3. Set-Top Box Design Example

The SoC implementation of a hypothetical set-top box shown in Fig. 1 is used as an example throughout this paper. The set-top box can simultaneously process two MPEG-2 video streams, which can either come from the RF-module or from the hard-disk, and can either be shown on a TV screen or saved to the hard-disk. A decryption unit allows to decrypt encrypted video streams. We assume that MPEG frames arrive periodically from the RF-module. When two MPEG streams are received simultaneously, the exact order of interleaved frame types is unknown. The set-top box can additionally process IP traffic and download web-content either to the screen or to the hard-disk.

We will focus on load analysis and response-time analysis for the system bus. We will assume *priority-based bus scheduling* of communication tasks sharing the system bus. We are going to demonstrate the improvement in analysis tightness that can be obtained when different types of system contexts are considered.

4. Intra Event Stream Contexts

Context-blind analysis assumes that in the worst-case, every scheduled task executes with its worst case execution

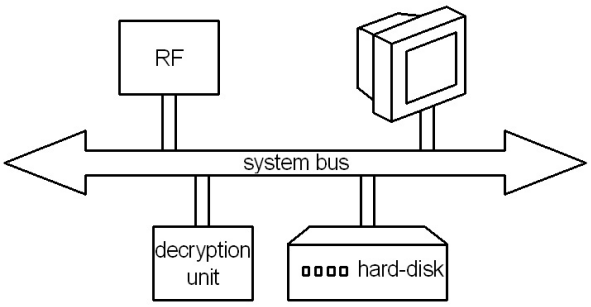

Figure 1. Set-top box example

time for each activation. In reality, different events often activate different behaviors of a computation task with different WCET, or different bus loads for a communication task. Therefore, a lower maximum load (and a higher minimum load) can be determined for a sequence of successive activations of a higher-priority task if the types of the activating events are considered. This in turn leads to a shorter calculated worst-case response time (and a longer best case response time) of lower-priority tasks. We call a sequence of different activating events an *intra event stream* context.

Mok and Chen introduced this idea in [1, 4] and showed promising results for MPEG-streams where the average load for a sequence of I-, P- and B-frames is much smaller than in a stream that consists only of large I-frames, which is assumed by a context-blind worst-case response time analysis [1]. However, the periodic sequence of types of activating events was supposed to be completely known.

In reality, intra event stream contexts can be more complicated. If no complete information is available about the types of the activating events, it is no longer possible to apply Mok's and Chen's approach. However, partial information may be available. Specifically, it may be possible to specify minimum and maximum conditions for the occurrence of each event type. In this case, a single worst-case and a single best-case sequence of events can be determined from the given min- and max-conditions that can be used to calculate the worst- and best-case load due to n consecutive activations of the task. n is an arbitrary integer value.

We have extended response-time calculation to exploit this idea. In the following, we show the worst-case calculation. Since min-conditions represent the lowest bound for the occurrence of an event-type in the sequence, our algorithm fulfills them first.

```
1  FulfillMinConditions(){
2    for(i = 0; i < MinConditionsList.size; i++){
3      MinCondition = MinConditionsList(i);
4      while(MinCondition not fulfilled in Sequence)
5        Sequence.add(MinCondition.type);
6    }
7    return Sequence
8  }
```

After this step, types have to be assigned to the remaining events to complete the sequence. This is reflected by the following method.

```
1  CompleteSequence(){
2    next = 0;
3    while(EventsToAdd >= 0){
4      type = WeightSortedEventTypeList.get(next);
5      Do as long as ((EventsToAdd >= 0) and
       (MaxConditions are not violated in Sequence)){
```

```
6         Sequence.add(type);
7         EventsToAdd = EventsToAdd - 1;
8       }
9       next = next + 1;
10    }
11    return Sequence;
12  }
```

Finally the resulting sequence is sorted by weight.

Intra event stream contexts can now be exploited for scheduling analysis. In case of a static priority preemptive scheduler, the following extension of equation 1 gives the worst-case response time for task i. $C_{j,k}$ is the WCET of the kth activation of task j in a worst-case sequence of activations due to an intra event stream context. If j is a task without intra event stream context information, $C_{j,k}$ equals C_j.

$$r_i \quad = \quad C_i + \sum_{\forall j \in hp(i)} \sum_{k=1}^{n_j(r_i)} C_{j,k} \qquad (2)$$

The code in section 2 has to be modified accordingly. Line 9 is replaced by the following call which calculates the 2nd sum in equation 2:

```
InterruptTime = MaxLoad(HighPriorTask,MaxActivations);
```

This method iterates through the weight-sorted sequence starting from the first event, adding up loads until the worst case load for n activations of the task has been calculated. If n is bigger than l, the sequence length, the method goes only through n mod l events and adds the resulting load to the load of the whole sequence multiplied by n div l.

```
1  MaxLoad(Task, n){
2    MaxLoad = 0;
3    for(i = 0;i <= n mod Task.Sequence.Length;i++){
4      MaxLoad =
       MaxLoad + Task.Sequence.get(i).getLoad;
5    }
6    MaxLoad = MaxLoad + Task.Sequence.Load *
     (n div Task.Sequence.Length);
7    return MaxLoad;
8   }
```

Let us apply this approach to our set-top box example. Suppose that the RF sends two multiplexed MPEG-2 video streams to the bus: the first video stream is displayed on the TV while the second is saved on the hard-disk. In each video stream, one of the following common MPEG-2 frame patterns is repeated: IBBPBB, IBPBPB or IBPB. In addition, the exact order of interleaved frame types in the multiplexed video stream is unknown. It is thus impossible to provide a fixed sequence of successive frame types in the multiplexed video stream. However, we can give minimum- and maximum-conditions for the occurrence of each frame type.

We determine min- and max-conditions for a multiplexed video stream with a length that is the smallest common multiple of the length of all patterns, 12 in this example. The following table shows the determined conditions.

frame type	min # of frames	max # of frames
I	2	4
P	2	4
B	6	8

We skip the details how to obtain these numbers. We assume that the size of each frame-type is fixed independent of the pattern that it appears in. Otherwise, we would simply specify more conditions.

The resulting partial sequence after fulfilling the min-conditions is IIPPBBBBBB. Worst-case frame types now have to be assigned to the remaining two frames to complete the sequence. I-frames generally have the largest size in MPEG-streams, B-frames have the smallest size. Since the upper bound on the occurrence of I-frames is 4 in our example, the remaining two frames have to be assumed to be I-frames. Therefore, the resulting sequence after this step is IIPPBBBBBBII. Finally, the sequence is sorted by weight (in this case frame size), resulting in IIIIPPBBBBBB.

Let us now observe the response-time analysis improvements through intra event stream contexts in our set-top box example. Suppose that in addition to the multiplexed video stream, IP traffic is sent via the bus to the hard-disk. We assume that the multiplexed video stream S_{mux} has a higher priority on the bus than the IP traffic S_{ip}. As an example, Fig. 2 shows for both context-blind (a) and intra event stream context (b) cases the calculated worst case response time of S_{ip} due to interrupts by S_{mux} for an IP traffic of 127, I-frame size of 106, P-frame size of 85 and B-frame size of 27. As can be seen, when considering the available intra event stream context information for the multiplexed video stream, gaps between successive executions of S_{mux} are larger than in the context-blind case. Since IP traffic is transmitted during these gaps (grey boxes in Fig. 2), a smaller number of interrupts of S_{ip} by S_{mux} is calculated considering intra contexts in comparison to the context-blind case. This leads to a reduction of the calculated worst-case response time of S_{ip}.

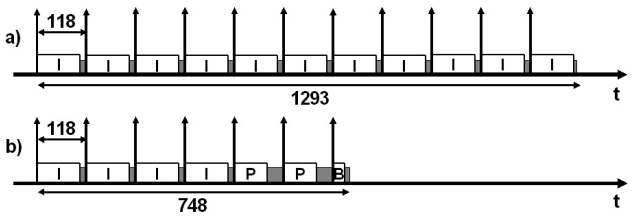

Figure 2. Worst-case response time calculation for IP traffic (grey boxes) a) without and b) with *intra* event stream context information

In Fig. 3, worst-case response time analysis improvements using intra event stream context information compared to the context-blind case are shown for S_{ip} as a function of the bus-speed. Typical frame size ratios are assumed for the multiplexed video streams. The I-frame size is normalized to 100 and the bus-speed is normalized to 1.

Curves **a** and **b** show the reduction of the calculated worst-case response time of S_{ip} for an IP traffic size of 120. Reducing the bus-speed increases the time needed to transmit one MPEG frame. This in turn leads to smaller gaps between successive transmissions of frames, leading to a larger number of interrupts of S_{ip}. The number of required gaps increases considerably faster in the context-blind case than in the intra event stream context case. Therefore, a greater reduction in the calculated worst-case response time

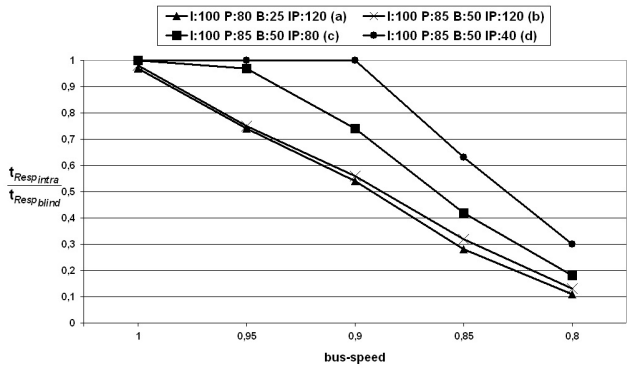

Figure 3. Improved worst-case response time analysis for IP-traffic due to *intra* event stream contexts

(up to 90 % in our example) is obtained in the intra event stream context case for slower bus speeds. This observation is important since the designer usually strives to reduce bus bandwidth as much as possible to save cost and power consumption.

Curves **c** and **d** show the reduction of the calculated worst-case response time of S_{ip} for smaller IP traffic sizes (80 and 40). When reducing the bus-speed, we observe generally a lower reduction of the calculated worst-case response time of S_{ip} for smaller IP traffic sizes. This is due to the fact that S_{ip} is interrupted less often by S_{mux} for smaller IP traffic sizes.

Another form of partial intra event stream context information are subsequences. Suppose that in our example, in addition to the existing min- and max-conditions we know that the subsequence [IIB] is always available in each sequence of twelve successive frames in the multiplexed video stream. Since the subsequence already assigns types to three frames, when evaluating the min- and max-conditions we only have to assign types to the remaining 9 frames keeping in mind the types already used in the subsequence. Therefore, some min-conditions are totally (min 2 I-frames out of 12 frames) or partially (min 6 B-frames out of 12 frames) fulfilled by the subsequence. The resulting sequence after fulfilling the remaining min conditions is [IIB]PPBBBBB.

Types still have to be assigned to two frames. Since the upper bound for the occurrence of I-frames is four (max 4 I-frames out of 12 frames), and two I-frames already exist in the sequence, the remaining two frames have to be assumed to be I-frames. Therefore the resulting sequence after this step is [IIB]IIPPBBBBB.

The 9 frames outside the subsequence are sorted by weight, however the complete sequence cannot be sorted. When executing `MaxLoad`, all insertion positions of the subsequence have to be tried to obtain the the worst-case sequence of length n.

5. Inter Event Stream Contexts

Context-blind analysis assumes that in the worst-case all scheduled tasks are activated simultaneously. In reality, activating events are often time-correlated, which rules out simultaneous activation of all tasks. This in turn may lead to a lower maximum number (and higher minimum number) of interrupts of a lower-priority task through higher-priority tasks, resulting in a shorter worst-case response time (and longer best-case response time) of the lower priority task.

Inter event stream contexts capture information about time-correlated events in different event streams in a way that can be exploited by performance analysis. Tindell introduced this idea for tasks scheduled by a static priority preemptive scheduler [8]. Each set of time-correlated tasks is grouped into a so called transaction. Each task is activated when an offset elapses after the activation of the transaction to which it belongs. Transactions are activated by periodic external events.

Tindell [8] showed that the worst-case contribution of a transaction to the response time of a lower-priority task occurs when the activation time of the lower-priority task happens as soon as possible after the *critical instant* of the transaction. The critical instant is the activation time of one of the transaction's higher-priority tasks. Subsequent activations of higher-priority tasks belonging to the transaction also have to happen as soon as possible after the critical instant.

Since all activation times of all higher-priority tasks belonging to a transaction are candidates for the critical instant of the transaction, the worst-case response time of a lower-priority task has to be calculated for all possible combinations of all critical instants of all transactions that contain higher priority tasks, to find the absolute worst-case.

The following `WorstCaseResponseTime` algorithm considers inter event stream contexts to compute the worst-case response time of a lower-priority task that doesn't belong to any transaction. If the lower-priority task itself belongs to a transaction, the interrupt-time by higher-priority tasks belonging to the same transaction has to be calculated separately.

```
1  WorstCaseResponseTime(LowPriorTask){
2    MaxResponseTime = LowPriorTask.WCET;
3    for each combination of critical instants
     determined by higer-priority tasks
     belonging to different transactions{
4      NewResponseTime = LowPriorTask.WCET;
5      Do{
6        OldResponseTime = NewResponseTime;
7        for each transaction{
8          Contribution = 0;
9          for each HighPriorTask of transaction{
10           MaxActivations = max activations of
             HighPriorTask during OldResponseTime
             considering HighPriorTask.Offset;
11           InterruptTime =
             MaxActivations * HighPriorTask.WCET;
12           Contribution =
             Contribution + InterruptTime;
13         }
14         NewResponseTime =
           NewResponseTime + Contribution;
15       }
21     }while(NewResponseTime > OldResponseTime)
22     if (NewResponseTime > MaxResponseTime)
23       MaxResponseTime = NewResponseTime;
24   }
25   return MaxResponseTime;
26 }
```

In comparison to the context-blind algorithm presented in section 2, the response time has to be calculated for all combinations of critical instants to get the worst-case response time (line 3). For each task in a transaction, the max

number of activations is calculated taking its offset into account (line `10`). For tasks not belonging to a transaction, instead of executing lines `7 - 15`, lines `5 - 10` of the context-blind algorithm in section 2are executed.

Let us apply Tindell's approach to our set-top box example. Suppose that the RF sends an encrypted MPEG-2 video stream to the bus. The decryption unit decrypts the stream, which we assume takes a fixed amount of time, and forwards it with the resulting time-offset via the bus to the TV. In addition, IP traffic is sent via the bus to the hard-disk. We assume that the encrypted video stream S_{enc} has the highest priority, the decrypted video stream S_{dec} has the middle priority and that the IP traffic S_{ip} has the lowest priority.

In order to show in isolation the analysis improvement due to inter event stream contexts, we will assume for now that all video-frames are I-frames. Since S_{enc} and S_{dec} are time-correlated, they are grouped into a transaction T.

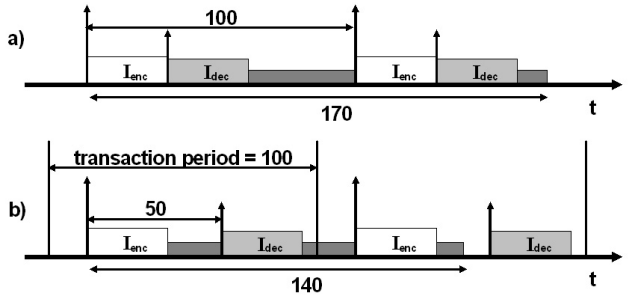

Figure 4. Worst-case response time calculation for IP traffic (grey boxes) a) without and b) with *inter* event stream context information

Let us observe the worst-case contribution of the transaction to the worst-case response time of S_{dec} and S_{ip}. As an example, Fig. 4 shows for both context-blind (a) and inter event stream context (b) cases the calculated worst case response time of S_{ip} and its interrupts through S_{enc} and S_{dec} for an IP traffic (grey boxes) of 50. The transaction-period is normalized to 100, the I-frame size is set to 30. As can be seen, when considering the available inter event stream context information for the video streams, one interrupt less of S_{ip} by S_{dec} is calculated. In general, when analyzing S_{ip},

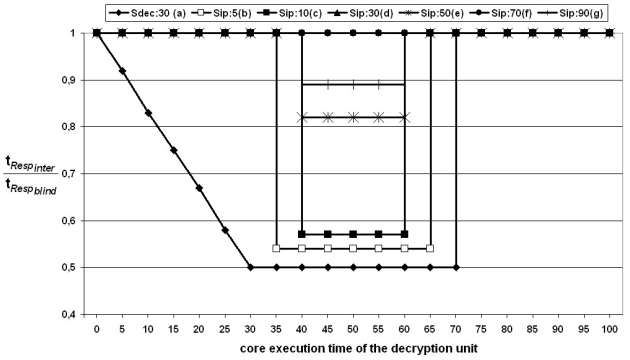

Figure 5. Improved worst-case response time analysis for IP-traffic and packet decryption due to *inter* event stream contexts

we have to try both critical points of transaction T, which are the activation times of S_{enc} and S_{dec}. In Fig. 5, analysis improvements with inter event stream context information in relation to the context-blind case are shown as a function of the offset between S_{enc} and S_{dec}, which is equal to the execution time of the decryption unit. The bus load due to the video streams is 60 %. Curve **a** shows the reduction of the calculated worst-case response time of S_{dec}. Depending on the offset, S_{dec} is either partially (offset value less than 30), completely (offset value more than 70) or not interrupted at all by S_{enc} (offset value between 30 and 70). The latter case yields a maximum reduction of 50 %.

Curves **b** - **g** show the reduction in the calculated worst-case response time of S_{ip} for different IP traffic sizes. The reduction is visible in the curves as dips. If no gaps exists between two successive executions of S_{enc} and S_{dec}, no worst-case response time reduction of S_{ip} can be obtained (offset value less than 30 or more than 70). If a gap exists, then sometimes one interrupt less of S_{ip} can be calculated (either through S_{enc} or S_{dec}), or there is no gain at all (curves **d** and **f**). Since the absolute gain that can be obtained equals the smaller worst case execution time of S_{enc} and S_{dec}, the relative worst-case response time reduction is bigger for shorter IP-traffic.

An important observation is that inter event stream context analysis reveals the dramatic influence that a small local change, in our example the speed of the decryption unit reading data from the bus and writing the results back to the bus, can have on system-performance, in our example the worst-case transmission time of lower-priority IP traffic. Systematically identifying such system-level influences of local changes is especially difficult using simulation due to the large number of implementations that would have to be synthesized and executed. On the other hand, formal performance analysis can systematically and quickly identify such corner cases.

6. Combination of Contexts

Inter event stream contexts allow to calculate a tighter number of interrupts of a lower-priority task through higher-priority tasks. *Intra* event stream contexts allow to calculate a tighter load for a number of successive activations of a higher-priority task. The two types of contexts are orthogonal: the worst-case response time of a lower-priority task is reduced both because fewer high-priority task activations can interrupt its execution during a certain time interval, and because the time required to process a sequence of activations of each higher-priority task is reduced. Therefore, performance analysis can be further improved if it is possible to consider both types of contexts in combination.

Let us apply the combination of event stream contexts to the scenario presented in the previous section, by additionally considering intra event stream contexts with the frame pattern IBBPBB. Therfore, both S_{enc} and S_{dec} have an intra and an inter context. In Fig. 6, we show analysis improvements considering both inter and intra event stream contexts in relation to the context-blind case as a function of the offset between S_{enc} and S_{dec}. Curve **a** shows the reduction of the calculated worst-case response time of S_{dec}. Since S_{dec} is interrupted at most once by S_{enc}, and the worst-case load

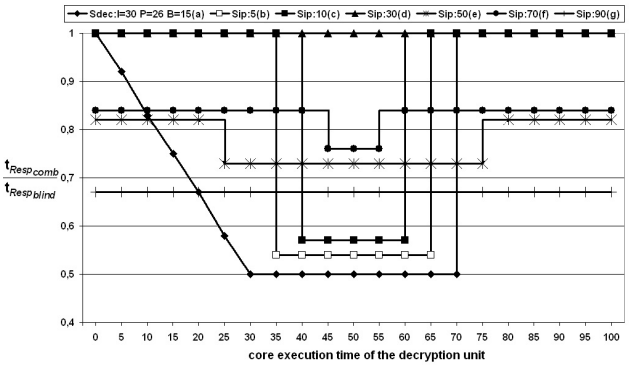

Figure 6. Analysis improvement due to the *combination of intra and inter* event stream contexts

produced due to one activation of S_{enc} is the transmission time of one I-frame, no improvement is obtained through the context combination in comparison to curve **a** in Fig. 5.

Curves **b** - **g** show the reduction of the calculated worst-case response time of S_{ip} for different IP traffic sizes. When comparing curves **b** and **c** (IP traffic sizes of 5 and 10) to curves **b** and **c** in Fig. 5), it can be seen that no improvement is obtained through the context combination. This is due to the fact that S_{ip} is interrupted at most once by S_{enc} and at most once by S_{dec}. Therefore, as in case **a**, the calculated worst-case load produced by the video streams is the same no matter whether the available intra event stream context information is considered or not.

Curve **d** shows that for an IP traffic size of 30, no improvements are obtained through the context combination in comparison to the *context-blind* case. This is due to the fact that for all offset-values, S_{ip} is interrupted exactly once by S_{enc} and exactly once by S_{dec}, and that the calculated worst-case load produced by the video streams due to one activation is the same no matter if intra event stream contexts are considered or not.

Curve **e** and **f** show that for IP traffic sizes of 50 and 70 improvements are obtained as a result of the context combination in comparison to both the intra and inter event stream context analysis. Let us focus on curve **e**. Since intra and inter event stream contexts are orthogonal, the reduction of the calculated worst-case response time of S_{ip} due to the intra event stream context is constant for all offset values. Since no reduction due to inter event stream context can be obtained for an offset value of 0 (equivalent to the inter event stream context-blind case), we are sure that the reduction shown in the curve for this offset value is only a result of the intra event stream context. On the other hand, the additional reduction between the offset values 25 and 75 is obtained due to the inter event stream context.

Curve **g** shows that for an IP traffic size of 90, even though the inter event stream context leads to an improvement (see curve **g** in Fig. 5), the improvement due to the intra event stream context dominates, since no dip exists in the curve. I.e. no additional improvements are obtained due to the context combination in comparison to the intra event stream context case.

7. Conclusion

In this paper we demonstrated that considering intra event stream contexts and inter event stream contexts can yield considerably tighter performance analysis bounds compared to a context-blind analysis. We used a priority-based system bus of a hypothetical set-top box as an example. We focused on realistic streams of bus load, where intra event stream contexts can be described only by minimum conditions, maximum conditions and possibly subsequences. We explored the analysis improvements for various bus speeds and various load distributions, by varying the speed of a decryption unit. The results show that the improvement due to intra event stream contexts is largest for high bus loads (up to 90 % in our example), which is a design point a designer is striving for. Considering inter event stream contexts improves analysis up to 50 % in our example.

Equally important, performance analysis reveals the dramatic influence that a small local change can have on system-performance. Systematically identifying such system-level effects due to local changes is especially difficult using simulation. On the other hand, the most complicated curves we presented took a couple of hundred milliseconds to compute on a standard desktop computer. These numbers show that formal performance analysis is ideal for design space exploration and corner case identification.

References

[1] S. K. Baruah, D. Chen, and A. K. Mok. Static-priority scheduling of multiframe tasks. In *Euromicro Conference on Real-Time Systems*, June 1999.

[2] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. DATE'03*, Munich, Germany, Mar. 2003.

[3] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Proceeding 40th Design Automation Conference*, Annaheim, USA, June 2003.

[4] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[5] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park, Colorado, USA, May 2002.

[6] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4), Apr. 2003.

[7] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *Proceedings 24th International Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, Dec. 2003.

[8] K. W. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Univ. of York, 1994.

[9] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.