Performance Analysis of Multimedia Applications using Correlated Streams

Kai Huang and Lothar Thiele*
Todor Stefanov and Ed Deprettere†

ABSTRACT

In modern embedded systems, data streams are often partitioned into separate sub-streams which are processed on parallel hardware components. To analyze the performance of these systems with high accuracy, correlations between event streams must be taken into account. No methods are known so far that are able to model such a scenario with the desired accuracy. In this paper, we present a new approach to analyze correlations and we embed this analysis method into a well-established modular performance analysis framework. The presented approach enables system-level performance analysis of complete systems by taking into account stream correlations and blocking-read semantics. Experimental results on a hardware-software prototyping system are provided that show the accuracy of the analysis in a practical application.

1. INTRODUCTION

For stream-oriented embedded applications the system design is moving from single processor implementations towards heterogeneous multi-processor System-on-Chip (MpSoC). These platforms are characterized by a large design space as there is a large degree of freedom in the partitioning of parallel application tasks, the allocation of concurrent hardware components, their binding to application processes, and the choice of appropriate resource allocation schemes. It is well acknowledged that design of such complex systems requires performance evaluation and validation techniques during the whole design trajectory. Because of the overall system complexity, fast estimation methods in an early design stage are critical for the exploration of large design spaces.

In order to obtain tight results in performance estimation, it is necessary to tackle the heterogeneity and complexity of the application and the underlying hardware/software platform as well as interferences and correlations between data streams, computation and communication. A typical example is the correlation between event streams and between computations in the Kahn Process model of computation [1] which is popular for modeling multimedia applications. For instance, in a typical split-join scenario, an input stream is split into several sub-streams and joined again after separate processing. Obviously, the sub-streams are highly correlated and an accurate performance analysis in terms of delays and buffer space usage requires (a) a model that is able to cover correlation between data streams and (b) a precise model of the blocking-read semantics and the associated blocking delay of processes.

*ETH Zürich, Switzerland, {huang,thiele}@tik.ee.ethz.ch
†University Leiden, Netherlands, {stefanov,edd}@liacs.nl

The system-level analysis of such heterogeneous systems with a high degree of internal correlations is currently mainly based on simulation. There are industrial simulation tools, e.g. Cadence's VCC [2], and academic tools, e.g. MPARM [3] and Ptolemy [4]. These simulation techniques allow the modeling of systems in any level of detail but they often suffer from long run times and from a high set-up effort for each new architecture, mapping and scheduling discipline. Worst-case bounds of system properties like throughput and end-to-end delay can not be obtained because of the inability to cover corner cases of the execution.

To achieve shorter run-times for simulation based methods, approaches that combine simulation and analysis have been proposed. In [5], a hybrid trace-based simulation methodology was proposed, and in [6] a method that combines the SystemC [7] based MPARM [3] simulation with an analytic technique [8]. Although these mixed methodologies can help to shorten the run-time of simulations, the problem of insufficient corner case coverage is still present.

Formal analytical methods, e.g. Symta/S [9], holistic analysis [10], timed automata [11], and modular performance analysis [12, 8], allow fast estimation speed and sufficient corner case coverage. Different methods have been developed to analyze systems in terms of their scheduling policies, the arrival event patterns of input streams, and the detailed modeling of processing and resource sharing. There are several approaches available to model event patterns and correlation *within* single event streams (e.g., the Syma/S framework [9]). However, due to the difficulty of exploiting timing correlations *between* event streams and blocking-read semantics, none of these existing frameworks for modular system level performance analysis is able to model them in the necessary level of detail. First steps in this direction have been presented in [13], [14], but the presented models tackle correlations between different event types and workloads only. In this paper, we present a method that is able to analyze the correlation between different streams while taking into account the blocking-read semantics of processing nodes. The contributions of this work can be summarized as follows:

- We present a methodology to model correlations in data streams and data distribution based on different types of delays, such as split delay, processing delay and blocking delay.
- The new model is embedded into the Modular Performance Analysis (MPA) framework of [12, 8].
- We show the applicability of the presented methods by analyzing a multimedia application and verifying the results by means of a hardware/software implementation on a fast prototyping platform.

2. MODULAR PERFORMANCE ANALYSIS

In the domain of embedded multimedia and digital signal processing applications, powerful abstractions have been developed to model and analyze the system performance. The framework used in this paper is an approach denoted as Modular Performance Analysis (MPA) [12]. The performance model of a system is composed of single abstract components that model (a) resources such as busses and processors, (b) event streams that are either communicated or triggering processes, and (c) resource sharing methods. The approach uses Real-Time Calculus [8] which itself is based on the theoretical framework called Network Calculus [15]. In particular, arrival curves $\alpha(\Delta)$, service curves $\beta(\Delta)$ and workload curves $\gamma(\Delta)$ [16] model certain timing properties of event streams, the capability of architecture elements, and the execution requirement of event streams, respectively, as shown in Fig. 1. Abstract components define the semantics of task execution and resource sharing in the system. To make the paper self-contained, a short description of these elements is given next. Workload curves $\gamma(\Delta)$ are not described here as their use is orthogonal to the new methods of this paper.

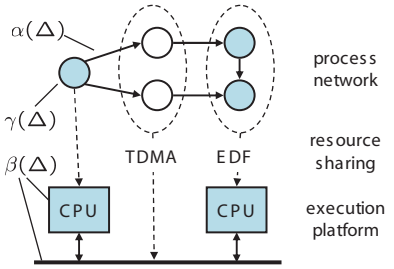


Figure 1: Elements of modular performance analysis.

2.1 Event Stream Model

All event streams in a system can be described using a cumulative function $R(s,t)$, defined as the number of events seen in the time interval $[s,t)$. While any R always describes *one* concrete trace of an event stream, a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of upper and lower *arrival curves* [17] provides an abstract event stream model that characterizes a whole class of (non-deterministic) event streams. $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ provide an upper and lower bounds on the number of events seen on the event stream in *any* time interval of length Δ, respectively:

$$\alpha^l(t-s) \leq R(s,t) \leq \alpha^u(t-s) \quad \forall s < t \tag{1}$$

with $\alpha^l(\Delta) = \alpha^u(\Delta) = 0$ for $\Delta \leq 0$. Arrival curves substantially generalize traditional event models such as sporadic, periodic, periodic with jitter, or any other arrival pattern with deterministic timing behavior. Therefore, they are suited to represent the complex characteristics of event streams in complex multiprocessor embedded systems.

2.2 Resource Model

In a similar way, the capability of a computation or communication resource can be described by a cumulative function $C(s,t)$, defined as the number of available resources, e.g. processor or bus cycles, in the time interval $[s,t)$. To provide an abstract resource model which models a whole set of possible resource behavior, we define a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower *service curves*:

$$\beta^l(t-s) \leq C(s,t) \leq \beta^u(t-s) \quad \forall s < t \tag{2}$$

with $\beta^l(\Delta) = \beta^u(\Delta) = 0$ for $\Delta \leq 0$. Again, service curves substantially generalize classical resource models as for example the bounded delay or the periodic resource model [18].

2.3 System Analysis

To analyze the performance of a concrete system, its essential properties need to be captured in an abstract performance model, which consists of a set of inter-connected abstract components. Abstract performance components model the application tasks in a system and define the semantics of how application tasks are executed on architecture elements.

For instance, consider a bus with bandwidth B that implements a TDMA protocol (Time Division Multiple Access). The TDMA individual slot length is denoted as s_i and the cycle length is denoted as $\bar{c}$, in which $\bar{c} = \sum s_i$. Then the service curves which a slot obtains can be modeled by:

$$\beta_i^l(\Delta) = B \cdot \min\{\lceil \Delta/\bar{c} \rceil \cdot s_i, \Delta - \lfloor \Delta/\bar{c} \rfloor \cdot (\bar{c} - s_i)\} \tag{3a}$$

$$\beta_i^u(\Delta) = B \cdot \max\{\lfloor \Delta/\bar{c} \rfloor \cdot s_i, \Delta - \lceil \Delta/\bar{c} \rceil \cdot (\bar{c} - s_i)\} \tag{3b}$$

Once the semantics of abstract components are defined, we can describe and analyze such a component using Real-Time Calculus [8]. For example, the semantics of a greedy system component can be described as follows: An incoming event stream, represented as a set of upper and lower arrival curves, flows into a FIFO buffer in front of a system component. The events trigger the instantiation of the corresponding application while being restricted by the availability of resources which are represented as a set of upper and lower service curves. The outgoing event stream can again be represented as a set of upper and lower arrival curves, while the remaining resource capacity can be represented as a set of outgoing upper and lower service curves. As has been shown [12], in this case the outgoing arrival curves α' can be determined as follows:

$$\alpha^{l'}(\Delta) = \min\{ \inf_{0 \leq \mu \leq \Delta} \{\sup_{\lambda > 0} \{\alpha^l(\mu + \lambda) - \bar{\beta}^u(\lambda)\} + \bar{\beta}^l(\Delta - \mu)\}, \bar{\beta}^l(\Delta)\} \tag{4a}$$

$$\alpha^{u'}(\Delta) = \min\{ \sup_{\lambda > 0} \{ \inf_{0 \leq \mu < \lambda + \Delta} \{\alpha^u(\mu) + \bar{\beta}^u(\lambda + \Delta - \mu)\} - \bar{\beta}^l(\lambda)\}, \bar{\beta}^u(\Delta)\} \tag{4b}$$

The abstract components are composed according to the flow of events and the use of resources. As they model subsystems including the resource sharing strategy, the modular analysis of a complex system is possible in terms of end-to-end delay, throughput and buffer sizes. For example, an upper bound of the maximum delay $d^{\max}$ experienced by an event and the maximal length $b^{\max}$ of the FIFO buffer at a greedy performance component is given by the following relation, see also [15]:

$$d^{\max} = \sup_{\lambda \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\} \right\} \tag{5}$$

$$b^{\max} = \sup_{\lambda \geq 0} \{\alpha^u(\lambda) - \beta^l(\lambda)\} \tag{6}$$

3. PERFORMANCE ANALYSIS OF CORRELATED STREAMS

In this section, we study stream correlations and methods to consider the corresponding effects in the performance analysis. To this end, we consider a split-join scenario, where a given event stream is distributed to several processing elements (split) and the results of the corresponding computations are combined later on (join), see Fig. 2. A typical example is the modified M-JPEG encoder [19]. Incoming video frames are fed into the encoder frame by frame. Each frame is partitioned into blocks using a certain distribution policy, blocks are compressed in parallel using concurrent hardware units. After compression, all compressed blocks join together to form a frame. Furthermore, we will distinguish between two different semantics of the split-process, namely the OR-semantics and the ORDER-semantics.

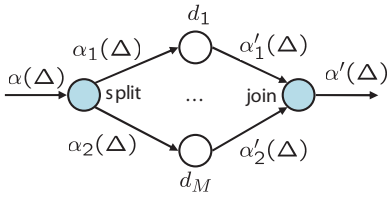


Figure 2: Process network representing the split-join scenario.

3.1 Join Process with OR-Semantics

A processing node with OR-semantics takes any event appearing on one of its inputs and processes all of them in a first-come-first-serve order. Without restricting the generality of our approach, we suppose that the join-process in the scenario of Fig. 2 just transfers input events to its output without any resource usage and in zero time. If there is additional processing necessary, this can easily be modeled by an additional process whose single input is connected to the output of the join-process.

Current analysis methods can not take into account the correlation between the streams $i = 1, ..., M$. After the partitioning of the given input stream they are considered as independent entities. As will be seen in the next example, this fact leads to a degraded accuracy of the performance analysis, i.e. delays and buffer sizes are overestimated to a large extent.

EXAMPLE 1. *Consider a simple split-join scenario in Fig. 2 with only two partitioned streams, i.e.* $M = 2$. *A simple TDMA scheme alternatively serves the two output streams with a fixed window size of* $10ms$. *A simple periodic input stream with* 2 *events per ms leads to the arrival curve* α *and the two output streams* α_1 *and* α_2 *shown in Fig. 3, left hand side. The two streams have their own processing routes and separate delays* d_1 *and* d_2, *respectively. Even in the most simple case with* $d_1 = d_2 = 0$, *the derived output curve* $\alpha' = \alpha_1 + \alpha_2$ *is overly pessimistic in comparison to the correct result* $\alpha' = \alpha$. *The shadowed part in the Fig. 3 shows the loss which was caused by previous analytical methods. For example, all subsequent subsystems will assume, that there are at most* 40 *events within a time interval of* $10ms$ *instead of just* 20.

The reason for the tremendous loss in accuracy can be seen in the fact that any information about time correlations is lost during the analysis. The following theorem leads to a method, which is able to consider time correlations between different event streams.

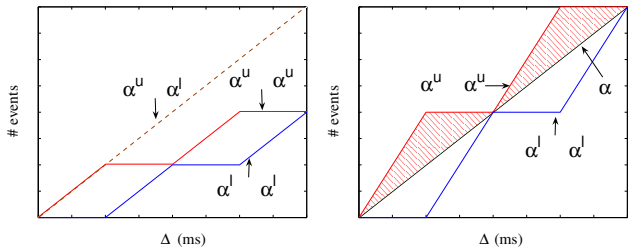


Figure 3: Arrival curves for the split-join-scenario.

THEOREM 1. *Assume an event stream with arrival curve* $\alpha = [\alpha^u, \alpha^l]$ *is split into* M *sub-streams which will be combined in a join process with OR-semantics. The delay of each substream between split and join is bounded by a tuple* $d_i = [d_i^{min}, d_i^{max}]$ *for all* $i \in M$. *Then the output of the join process is an event stream that can be bounded by the arrival curve*

$$\alpha'^u(\Delta) = \min\left\{\sum_{i=1}^{M} \alpha_i^u(\Delta + \delta_i), \alpha^u(\Delta + \delta^{max} - \delta^{min})\right\} \tag{7a}$$

$$\alpha'^l(\Delta) = \max\left\{\sum_{i=1}^{M} \alpha_i^l(\Delta - \delta_i), \alpha^l(\Delta + \delta^{min} - \delta^{max})\right\} \tag{7b}$$

where $\delta_i = d_i^{max} - d_i^{min}$, $\delta^{max} = \max_{i \in M}\{d_i^{max}\}$, *and* $\delta^{min} = \min_{i \in M}\{d_i^{min}\}$.

PROOF. Because of space limitations, we will only prove a simplified version of the above theorem which considers only two streams, one stream has delay $d_1 = d$ and the other one $d_2 = 0$. Extensions to the above theorem are straightforward. Let us look at the event stream arriving at the input. The split process selects events that are transferred to stream 2 and then appear without any delay at the output stream. $\alpha'^u(\Delta)$ is the maximum number of events in some interval of length Δ and let the right interval in Fig. 4 represent such an interval in the input stream. Then the input events transferred to stream 1 that are appearing in the same time output window are represented as the left interval. Then one can conclude that both, $\alpha^u(\Delta + d)$ and $\alpha_1^u(\Delta) + \alpha_2^u(\Delta)$ are lower bounds of $\alpha'^u(\Delta)$. In a similar way, $\alpha^l(\Delta + d)$ and $\alpha_1^l(\Delta) + \alpha_2^l(\Delta)$ are upper bounds of $\alpha'^l(\Delta)$. □

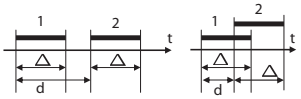


Figure 4: Illustration for the proof of Theorem 1.

3.2 Join Process with ORDER-Semantics

In the area of modeling stream-oriented data processing applications, the Kahn Process Network (KPN) model of computation [1] is popular for modeling parallel systems. The KPN model assumes a network of concurrent

autonomous processes that communicate in a point-to-point fashion via unbounded FIFO channels, using a blocking-read and nonblocking-write communication primitives. Given these properties, a KPN is determinate, which means the same input/output relations hold irrespective of the timing and scheduling policies.

In the OR-semantics, an abstract performance component is modeled as a greedy processing block that reads input data whenever they are available at any of its inputs. Thus the ordering information of events within different sub-streams is lost, which leads to re-ordering of the output blocks. This not only contradicts the KPN semantics but also leads to an undesirable reordering of blocks in an image frame, if we consider again the M-JPEG encoder example.

Again, we assume that the *split* and *join* processes are infinitly fast. Besides the delays d_i considered in Section 3.1, we now have to consider the blocking-delays bd_i at the join process for each stream i.

Let us first define the minimal and maximal split interval $sd_{ij}^{\min}$ and $sd_{ij}^{\max}$ which are the minimal and maximal relative time differences between an event in stream i and the most recent one in stream j, see also Fig. 5. This delay depends on the distribution policy of the *split process*, i.e., different distribution policies cause altered delays. If only the split distances $se_{ij}^{\min}$ and $se_{ij}^{\max}$ *in the number of intermediate events* between the different streams are known, then one can use the input arrival curve $\alpha(\Delta)$ in order to compute the corresponding delays. For example, $sd_{ij}^{\min} = \inf\{\lambda \geq 0 \mid \alpha^u(\lambda) \geq se_{ij}^{\min} + 2\}$, $sd_{ij}^{\max} = \inf\{\lambda \geq 0 \mid \alpha^l(\lambda) \geq se_{ij}^{\max} + 1\}$.

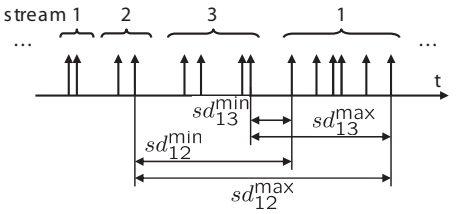


Figure 5: Illustration of the split interval.

As an example, let us consider a periodic input stream with event rate $\bar{r}$ and a TDMA split process that assigns windows of size s to each stream in the order $1, 2, ...M, 1, 2....$ If s is a multiple of the distance between two events $1/\bar{r}$, then

$$sd_{ij}^{max} = s \cdot (i-j) \bmod M \quad sd_{ij}^{min} = sd_{ij}^{max} + 1/\bar{r} - s \quad (8)$$

The second part of the delay chain we need to consider consists of the minimal and maximal accumulated processing delays $pd_i^{\min}$ and $pd_i^{\max}$ of the individual streams i, including the time spent in the buffers in front of the computing resources, see also the 'blank' nodes in Fig. 2. One can apply (5) and (6) in order to obtain upper bounds on this delay and the corresponding backlog in front of the computing or communication resource. For more complex operation chains, the whole method of modular performance analysis can be used in order to determine lower and upper bounds on the processing delays.

The third part we need to consider is the blocking-read delay bd_i which occurs if in-order output is required. Due to the blocking-read semantic of the *join process*, an event of a sub-stream is read by the join process only after all previous events (in terms of the original order of the events in the input stream) that are still being processed in other sub-streams have been read. An upper bound on the blocking-read delay can be determined using the split interval and processing delay as defined above:

$$bd_i^{\max} = \max_{j \in M \wedge j \neq i} \{0, pd_j^{max} - pd_i^{min} - sd_{ij}^{min}\} \quad (9a)$$

The blocking-read delay can be used for the calculation of the backlog in front of the *join process*. We can obtain an upper bound of the number of waiting events as

$$b_i^{\max} = \alpha_i^{'u}(bd_i^{max}) \quad (10)$$

where $\alpha_i^{'}$ is the arrival curve of a processing route in front of the *join process*, see Fig. 2.

EXAMPLE 2. *Again consider the scenario in Example 1. The input event stream α is split into two sub-streams, namely α_1 and α_2. The event e_2 of α is dispatched to sub-stream α_1 at time t_2 and its minimum processing delay is pd_1^{min}. The closest previous event e_2 which has been put to α_2 has maximum processing delay pd_2^{max} and will reach the* join process *at time t_4. Then the maximum blocking-read delay bd_1^{max} which e_2 has to wait in the buffer queue in front of the* join process *is $t_4 - t_3$.*

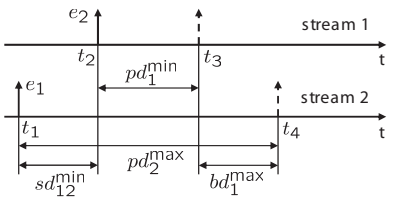


Figure 6: Illustration of the maximum blocking-read delay.

Combining all the information, the final end-to-end delay of each sub-stream from the output of the split to the output of the join process can be calculated as follows:

$$d_i^{min} = \max_{j \in M \wedge j \neq i} \{pd_i^{min}, pd_j^{min} - sd_{ij}^{max}\} \quad (11a)$$

$$d_i^{max} = \max_{j \in M \wedge j \neq i} \{pd_i^{max}, pd_j^{max} - sd_{ij}^{min}\} \quad (11b)$$

These values can now be used in (7a) and (7b) in order to determine the resulting arrival curve $\alpha'(\Delta)$.

4. EXPERIMENTAL RESULTS

In this section, we apply our method to analyze a real-life system, namely a modified M-JPEG encoder [19]. We derive the output arrival curves and the final end-to-end delay and compare them with the results obtained from previous methods. The analytic performance analysis has been done using the MPA toolbox, see http://www.mpa.ethz.ch.

Like traditional M-JPEG encoders, the modified M-JPEG encoder compresses a sequence of frames by applying JPEG compression to each frame. Because of the inherent parallelism in the JPEG algorithm, a frame can be split into macro blocks which can be compressed in parallel by concurrent hardware components. Our application scenario shown in Fig. 7 contains four processors and two dedicated hardware IP-Cores, which are interconnected by a bus with a TDMA arbitration scheme. The *split CPU* splits image

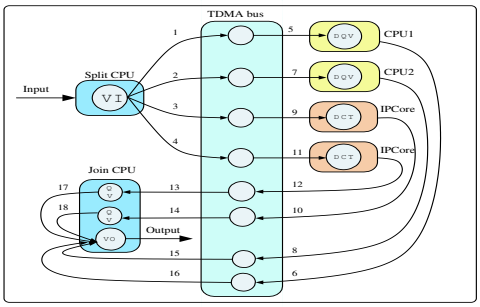


Figure 7: M-JPEG encoder scenario.

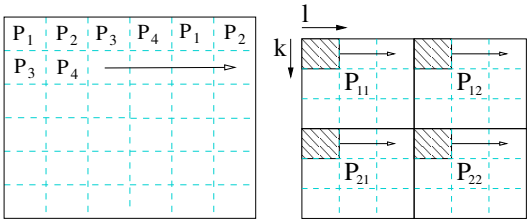
frames into macro blocks, scans macro blocks row by row, then dispatches them to four different processing routes. In two of the routes, the generic processing elements *CPU1* and *CPU2* conduct a software compression to incoming macro blocks whereas in the other two routes dedicated hardware IP-Cores perform a hardware compression. The compressed macro blocks assemble in the *join CPU* to form a stream of JPEG-encoded frames.

For comparison, we prototyped this M-JPEG encoder using ESPAM [20] which is a fast hardware/software prototyping environment using a KPN model compiler and a Field Programmable Gate Array (FPGA) platform. The target system runs on a set of XILINX VIRTEXII-PRO FPGA. The specifications of the involved pieces of hardware are listed in Table 1. Microblaze soft processors are used as general purpose processing elements for *CPU1*, *CPU2*, and *JoinCPU*. The *IP-Core* conducts a hardware compression with constant delay for the processing of each macro block. The *TDMA bus* has 6 slots with a slot length of 2048 cycles. One input frame contains 128 macro blocks, and one macro block consists of 8×16 pixels (256 32-bit words).

Table 1: Hardware Specifications.

Hardware	Frequency [MHz]	Throughput [bytes/cycle]	other features
TDMA bus	100	4	6 slots, 2048 cycles each
IPCore	100	4	94-stages pipeline
Microblaze	100		

For design space exploration, we conducted three experiments by applying two different distribution policies to the *split CPU*, i.e. modulo distribution and block distribution. In case of the modulo distribution, macro blocks are dispatched to the four routes alternately, as shown in Fig. 8(a). We can compute the split intervals using (8) with $M = 4$. In case of the block distribution, an image frame is split into four zones, as depicted in Fig. 8(b). Macro blocks inside one zone are dispatched to one processing route. Since the *split CPU* scans blocks always row by row, split interval calculation becomes more involved. As shown in Fig. 8(b), We associate the stream i with a tuple $[k_i, l_i]$, i.e. $P_{k_i l_i}$ represents the stream i. Now the split intervals can be determined as follows: when $k_i = k_j$, the delay is bounded by (12a) and (12b); when $k_i > k_j$, the delay is bounded by (12c) and (12d); when $k_i < k_j$, we assume positive infinity, because $k_i < k_j$ represents the interval between two frames. Inside (12), N_1 and N_2 denotes the number of macro blocks in a row and a column, respectively. M denotes the number of zones in each row and column (i.e. M^2 is the total number of zones). $\bar{r}$ represents the input block rate.



(a) Modulo distribution. (b) Block distribution.

Figure 8: Block distribution policies.

$$sd_{ij}^{min} = \left(1 + \frac{N_1}{M} \cdot ((l_i - l_j) \bmod M - 1)\right) \cdot \frac{1}{\bar{r}} \tag{12a}$$

$$sd_{ij}^{max} = \left(\frac{N_1}{M} \cdot ((l_i - l_j) \bmod M)\right) \cdot \frac{1}{\bar{r}} \tag{12b}$$

$$sd_{ij}^{min} = \left(1 + (M + l_i - l_j - 1)\frac{N_1}{M} + (k_i - k_j - 1)\frac{N_1 \cdot N_2}{M}\right) \cdot \frac{1}{\bar{r}} \tag{12c}$$

$$sd_{ij}^{max} = \left((l_i - l_j)\frac{N_1}{M} + (k_i - k_j)\frac{N_1 \cdot N_2}{M}\right) \cdot \frac{1}{\bar{r}} \tag{12d}$$

The information given above is sufficient to use the framework described in this paper. We use the Modular Performance Analysis (MPA) combined with (11) and (7) to derive the output arrival curves and the end-to-end delay. The final results are shown in Figs. 9, 10, and 11.

In the first experiment, we evaluate the modeling of nodes with Or-semantics using modulo distribution policy. In Fig. 9, we show analytical results of both Or-semantics and previous methods with an input frame rate *6 frames/s*. The solid lines represent curves obtained by the new method, taking into account the correlations between event streams. The dotted lines represent curves generated by previously available methods. We can observe that the new presented analysis method leads to considerably tighter analytic bounds. The maximum intervals between two output macro blocks are also depicted in the figure. The new obtained maximum interval(*3.46 ms*), is above 2 times less than the previous *7.63 ms*. The calculated worst case end-to-end delay of one macro block in both cases is *3.44 ms*. In addition, we compared the analytical results to those obtained from measurements on the prototyping system. From the traces generated by the FPGA implementation, the maximum block interval is *2.61 ms* and the average end-to-end delay of amacro block is *2.14 ms*. Compared to the measurement results, the calculated maximum block interval and the worst case end-to-end block delay are valid and in acceptable ranges.

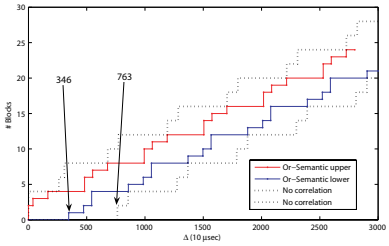


Figure 9: Or-semantics and no correlation in modulo distribution experiment.

In the second experiment, we compare the results between OR-semantics and ORDER-semantics of processing nodes with modulo distribution policy. We increase the input frame rate to *9.6 frames/s* in order to amplify the blocking

read behavior. Now the system is dominated by blocking-read delays in front of the *join* processor. This fact can be seen from Fig. 10. The curves corresponding to a join node with OR-semantics (solid lines) do not take into account the delays caused by the blocking read operation (in-order assembly of macro blocks). The *join* processor will read whatever data are available at any of its input. Thus we get too optimistic results, as shown by the tighter bounds in the figure. Once the ORDER-semantics is taken into account, the bounds become weaker as the differences between the minimal and maximal delay of events grow substantially. Because of the analytical approach, the bounds correctly correspond to the worst case behavior of the multiprocessor system. The results can be used to obtain bounds on jitter, maximum throughput, worst-case and best-case delay and buffer sizes.

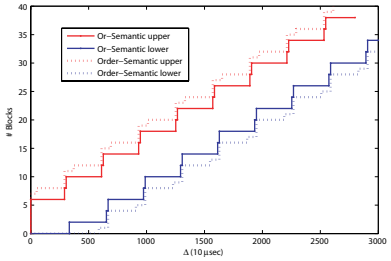


Figure 10: Or-semantics and Order-semantics with modulo distribution.

In the third experiment, we compare the block distribution and modulo distribution with Order-semantics. Again, we set the input frame rate to *6 frames/s*. From Fig. 11, the curves corresponding to the block distribution policy (dotted lines) are pessimistic. The reason is that the block distribution yields burst events to the Microblaze processor and leads to a extremely large worst case delay for some blocks. This worst case delay will be applied to every block because our analytic method deals with worst case analysis. Compared to the measurement, the result obtained from this experiment is still pessimistic. We are currently developing new method to improve the bounds.

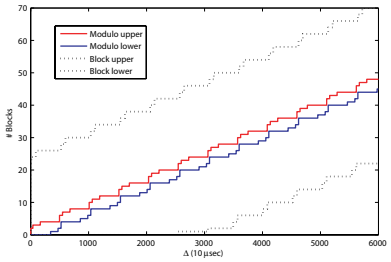


Figure 11: Block distribution and modulo distribution with Order-semantic.

5. CONCLUSION

In this paper, we present a new method for analyzing correlated data streams and blocking-read semantics in the case of data distribution. We embed this method into the Modular Performance Analysis (MPA) framework of [12, 8] and prove the applicability of the presented methods by analyzing a real-life application. The analytic performance analysis has been done using the MPA toolbox, see http://www.mpa.ethz.ch. In the future, we will extend the basic concept of analyzing correlations towards more complex scenarios.

Acknowledgements

This research has been funded by European Integrated Project SHAPES under IST Future Emerging Technologies - Advanced Computing Architecture (ACA). Project number: 26825.

6. REFERENCES

[1] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP Congress 74*, North-Holland Publishing Co., 1974.

[2] "The Cadence Virtual Component Co-design (VCC)." `http://www.cadence.com/products/vcc.html`.

[3] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing on-chip communication in a MPSoC environment," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, pp. 752–757, IEEE Computer Society, 2004.

[4] E. Lee and A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, 1998.

[5] K. Lahiri, A. Raghunathan, and S. Dey, "System level performance analysis for designing on-chip communication architectures," *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, vol. 20, no. 6, pp. 768–783, 2001.

[6] S. Künzli, F. Poletti, L. Benini, and L. Thiele, "Combining simulation and formal methods for system-level performance analysis," in *Proc. Design, Automation and Test in Europe (DATE)*, March 2006.

[7] "The Open SystemC Initiative (OSCI)." `http://www.systemc.org`.

[8] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. 101–104, 2000.

[9] K. Richter and R. Ernst, "Event model interfaces for heterogeneous system analysis," in *Proc. 5th Design, Automation and Test in Europe (DATE)*, p. 506, IEEE Computer Society, March 2002.

[10] P. Pop, P. Eles, Z. Peng, V. Izosimov, M. Hellring, and O. Bridal, "Design Optimization of Multi-Cluster Embedded Systems for Real-Time Applications," in *Design, Automation and Test in Europe (DATE 2004)*, pp. 1028–1033, 2004.

[11] T. Amnell, E. Fersman, L. Mokrushin, P. Petterson, and W. Yi, "Times - a tool for medlling and implementation of embedded systems," in *TACAS 02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, p. 460, 2002.

[12] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proc. 6th Design, Automation and Test in Europe (DATE)*, pp. 190–195, March 2003.

[13] M. Jersak, R.Henai, and R. Ernst, "Context-aware performance analysis for efficient embedded system design," in *Proc. Design, Automation and Test in Europe (DATE)*, March 2004.

[14] E. Wandeler and L. Thiele, "Characterizing Workload Correlations in Multi Processor Hard Real-Tim e Systems," in *11th IEEE Real-Time and Embedded Technology and Applications Sympo sium (RTAS)*, pp. 46–55, March 2005.

[15] J. Le Boudec and P. Thiran, *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

[16] A. Maxiaguine, S. Künzli, and L. Thiele, "Workload characterization model for tasks with variable execution demand," in *Proc. 7th Design, Automation and Test in Europe (DATE)*, 2004.

[17] R. Cruz, "A calculus for network delay," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114–141, 1991.

[18] I. Shin and I. Lee, "Compositional Real-Time Scheduling Framework," in *Proceedings of the Real-Time Systems Symposium (RTSS)*, pp. 57–67, IEEE Press, 2004.

[19] T. Stefanov, "Converting weakly dynamic programs to equivalent process network specifications," Sept. 2004. Ph.D. dissertation book, Leiden University, Leiden, The Netherlands, September 2004, ISBN: 90-9018629-8.

[20] H. Nikolov, T. Stefanvo, and E. Deprettere, "Multi-processor system design with espam," in *4th IEEE/ACM/IFIP Int. Conf. on HW/SW Codesign and System Synthesis (CODES-ISSS'06)*, Oct. 2006.