Probabilistic Performance Risk Analysis at System-Level*

Alexander Viehl[1], Markus Schwarz[1], Oliver Bringmann[1], Wolfgang Rosenstiel[1,2]

[1]FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
{viehl,schwarz,bringman}@fzi.de

[2]Universität Tübingen
Sand 13
72076 Tübingen, Germany
rosenstiel@informatik.uni-tuebingen.de

ABSTRACT

We present a novel hybrid approach for performance analysis of a system design. Unlike other approaches in this area, in this paper we do not focus on the determination of pessimistic best-case and worst-case quantities of system properties. Our proposed analysis methodology determines qualitative numbers between best-case and worst-case of system properties and quantifies them with probabilities. For this issue, we combine local coarse-grained profiling and formal system-level analysis models in a hybrid approach for an early quantitative determination of qualitative system properties. Our approach considers the control-flow of communicating processes and the impact of blocking communication instances on the temporal behavior of the entire system during formal analysis. This can be used for determining the global system performance. The application of our new methodology leads to an inclusion of probabilities concerning system properties and allows an early performance risk estimation of a design with regard to predefined system requirements and constraints.

Categories and Subject Descriptors: B.8.2 [Performance Analysis and Design Aids] G.3 [Probability and Statistics]: Stochastic Processes

General Terms: Performance, Measurement

Keywords: Performance Analysis, Probabilistic Risk Quantification

1. INTRODUCTION

The gap between complexity growth and increasing time-to-market pressure is continuously growing. Several approaches try to close the gap by abstracting system properties in an automated design process. Early available design characteristics are essential for designers to validate a design concerning requirements. Whereas different approaches exist to incorporate worst-case and best-case properties of the system, the intervals (for e.g. performance) are often too pessimistic and far away from the real system behavior. Moreover, rare cases that appear only once like initial cache misses can expand the intervals of present quantitative best-case or worst-case analysis too much. Therefore, a resulting system architecture with regard to these worst-case analyses is quite often overdimensioned. Although this might be acceptable for real-time critical systems, the designer does not know about the typical quantities of qualitative properties at an early design stage. The amount of cases that violate the requirements is unknown. Figure 1 depicts a typical scenario in which the requirement for a system property is between the best-case and the worst-case. Without any quantitative characterization of the qualitative system property, the designer does not know, how many cases are critical and not met. The risk for property m (e.g. end-to-end latency or buffer utilization) in this scenario is characterized by the sum of the probabilities $P(m_i)$ of all cases m_i between the worst-case $wc(m)$ and the requirement $req(m)$ of the system property.

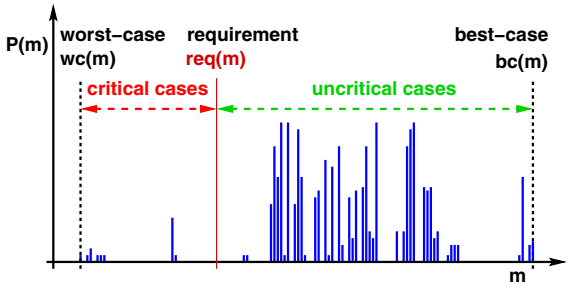


Figure 1: Quantitative Evaluation

Estimations are often sufficient for an early evaluation of the design properties. We apply coarse-grained profiling for determining the execution time of communication free parts of a system model. This architectural mapping process is briefly described in Section 3. The resulting quantities are incorporated in an analytic system model, the communication dependency graph (CDG) presented in Section 2.1. Using this graph, we are performing system-level analysis to determine qualitative statements about the system behavior. Qualitative means that we identify possible cases between best-case and worst-case. These cases are quantified with probabilities and their interrelation is considered by our analysis approach. Our novel methodology tries to combine the advantages of both worlds: We derive local execution time distributions by profiling dedicated components and we perform formal communication analysis to explore the global module interaction with respect to execution time profiles between best-case and worst-case bounds. The remainder of the paper is organized as follows: Using the execution times of the activity model that is presented in Section 2.1, we calculate idle times. These idle times are represented by discrete probability distributions of time the control-flow of the processes has to wait in blocking communication endpoints. Our methodology for idle time determination is called communication analysis and is presented in Section 4. The derived quantified qualitative information can further be used for an exploration of the underlying system architecture. Our evaluation methodology is presented in Section 5. We applied our methodology for qualitative design evaluation on a real design of a Viterbi decoder. The results are shown in Section 6.

2. STATE OF THE ART

Approaches that only rely on best-case/worst-case assumptions are very often too pessimistic for real scenarios. Therefore, related work on performance analysis has to be evaluated with re-

*This work was partially supported by the BMBF project URANOS under grant 01M3075D and by the DFG project "Communication Analysis for Network-on-Chip" under grant BR 2321/1-1

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria

spect for their capability of extending these worst-case/best-case assumptions.

Timed Petri Nets [24] are able to represent the internal behavior of a system. Although there exist stochastic extensions by generalized stochastic Petri nets (GSPN) [14], these do not consider execution times of the real system components. Furthermore, synchronization by communication and the specification of communication types have to be modeled explicitly and can not be extracted from executable functional specifications like a SystemC model of a design.

System-level performance and power estimation based on Stochastic Automata Networks (SAN) are introduced in [13]. The system including probabilities of execution times is modeled explicitly in SAN. The real execution behavior of the components related to timing and control-flow of a functional implementation is not considered.

A method for analyzing the worst case timing behavior of concurrent systems is based on communicating automata [20], an extension to timed automata [3, 6]. However, these models rely on totally synchronous communication. This restriction reduces the possible degree of concurrency and is not realistic for communication with buffering and latencies. Stochastic automata [7] extend the model by general probability distributions to verify the performance of systems. However, the system has to be modeled explicitly and no bottom-up evaluation of a functional system model is given.

In [25], the authors abstract from the internal processes and operate on an acyclic task graph. This model is based on the assumption that each task has a statically determined execution time and each task starts its execution once all input signals are available. The main drawback of this model is the missing support of conditional control structures. Therefore, the basic task graph model is extended by adding control dependencies [16, 23]. Due to the acyclic structure, the modeling of different communication protocols and data-dependent loops is not allowed. The model was extended by a stochastic characterization of execution time probabilities [12]. The internal control-flow of the processes, communication protocols and synchronization primitives are still not considered. No method for deriving the abstract model from real implementations is given. Recent efficient approaches address general hardware/software platforms, with respect to user-specified I/O event models [17, 8, 2]. The main issues are caused due to the focus on the I/O stream behavior and the missing consideration of the control-flow of the processes. Furthermore, no quantified qualitative properties between upper and lower bounds of the system characteristics are determined. In [1] an analysis method was presented that allows the extraction of hierarchical event streams out of the control-flow graph of a system. The analysis focuses the determination of worst-case system properties based on best-case module event analysis. An enhancement for reducing pessimism in worst-case performance analysis was presented in [21]. Execution traces are incorporated to determine tighter bounds of performance properties. The model is still abstract and does not consider the internal control-flow of processes as well as it focuses on worst-case properties.

Improvements on WCET analysis of software processes for tightening the bounds of resulting intervals by incorporating dependencies and correlations between code fragments have been presented in [5, 10]. Although these methods are promising for the analysis of single software tasks, they are not applicable on systems with complex interaction and communication. Furthermore, they only focus the determination of WCET and not the behavior between worst-case and best-case. On the other hand, profiling is used for the determination of characteristics like execution time, resource utilization or memory requirements. This approach is very fine-grained and slow. Transaction Level Modeling (TLM) uses models with components on different levels of abstraction for speeding up simulation with a potential loss of accuracy. Profiling or simulating a complete system of parallel processes consumes a lot of computational resources and synchronization overhead for coupling multiple simulators is an issue. A modification of the architecture or the functionality of a component requires to analyze the entire system again [22, 9].

2.1 Activity Model

To represent the temporal behavior of a system, a model called communication dependency graph (CDG) is used. It was originally developed for formal communication analysis by determining communication instances that synchronize the control-flow of communication partners [18]. We apply this model for the system-level representation of a design. The model considers temporal best-case and worst-case properties of communication and computation properties. We extend the model in Section 3 for quantitative determination of the qualitative design properties.

A *communication dependency graph* (CDG) denotes a consolidated representation of a system consisting of communicating processes. In each process, only communication endpoints and the temporal and causal behavior between them is considered. The control-flow is represented by edges connecting communication endpoints. An edge exists if at least one path in the control-flow graph connects the communication endpoints without passing any other communication endpoint. The communication endpoints are characterized concerning the synchronization behavior, and whether they represent sending or receiving events. A CDG is basically denoted by

$CDG := < V_{CDG}, E_{CDG}, E_{COM}, \tau_{CDG}, l_{CDG} >$, where

- V_{CDG} is a set of nodes representing communication endpoints.
- $E_{CDG} \subseteq V_{CDG} \times V_{CDG}$ is a set of directed edges describing the precedence dependencies between nodes.
- $E_{COM} \subseteq V_{send} \times V_{rec}$, with $V_{send} = v \in V_{CDG} : \tau_{CDG}(v) \in \{send_{async}, send_{sync}\}$ and $V_{rec} = v \in V_{CDG} : \tau_{CDG}(v) \in \{receive_{async}, receive_{sync}\}$ is a set of directed edges describing the communication instances
- The function $\tau_{CDG}(v)$: $V_{CDG} \rightarrow \{send_{async}, send_{sync}, receive_{async}, receive_{sync}\}$ denotes the type of each node.
- The edge weights are represented by the function $l_{CDG} : E_{CDG} \rightarrow \mathbb{N}_0 \times \mathbb{N}_0$ with minimum and maximum execution time $l_{CDG}(v_1, v_2) = (cs_{min}, cs_{max})$ between the nodes $v_1, v_2 \in V_{CDG}$.

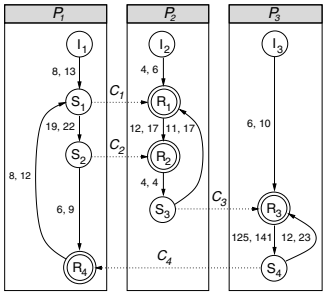


Figure 2: Communication Dependency Graph

A CDG example consisting of three processes P_1, P_2, and P_3 is depicted in Figure 2. The processes communicate via asynchronous communication instances C_x with non-blocking sender and blocking receiver nodes.

3. ARCHITECTURAL MAPPING

In this section, we briefly explain our mapping and exploration flow depicted in Figure 3. The starting point is a functional SystemC [15] model of a design, a platform description, mapping information, and the environment of the system. Although we are focusing on mapping of software processes onto microcontrollers in this paper, our approach is applicable for a hardware realization of the functionality as well. The SystemC model is simulated together with a specified environment. The environment defines the interaction of the world with the model, (e.g. sensor packets for an automotive control system) and the period with jitter between the packets. We use introspection for gaining access to the communication data sent between SystemC processes. As results, we determine temporally ordered communication calls of each process and the data sent through each communication channel.

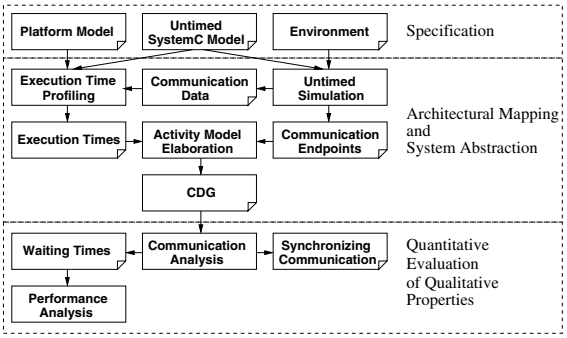


Figure 3: Mapping and Analysis Flow

We use the ordered communication calls for elaborating a consolidated control-flow as it is represented by E_{CDG} and V_{CDG} in each CDG process. The communication data are used as input for determining the execution time of code fragments on the target platform with regard to the data computed by the design originating from the environment. For performing execution time profiling, we have to map each process from the original SystemC model to the target platform. Therefore, we are using the tool PORTOS [11] to generate a set of C/C++ files, each including the functionality of one SystemC process. The access to SystemC ports is mapped to special `read(...)` and `write(...)` calls as depicted in Figure 4a.

After compilation, each single process is executed on an Instruction Set Simulator (ISS). Therefore, we extended the SimpleScalar [4] ISS to handle the special syscalls contained in the communication stubs. When a communication function is called, the special syscall is recognized by our extension to the ISS. Two timestamps are taken that encapsule the computation time of the communication routine. Using the first timestamp of communication instance C_i and the last timestamp of the previous communication instance C_{i-1}, the computation time between these two subsequent communication instances of one process is calculated. In Figure 4a, the computation time between receiving C_i and sending C_{i+1} is calculated by $t_3 - t_2$.

In the next step, the collected execution times are combined with the elaborated control-flow to the activity model, the CDG.

Based on the CDG, communication analysis is performed as described in Section 4 to determine the impact of blocking communication on the global temporal behavior of the entire system. Further results are the detection of non-synchronizing communication instances that indicate need for buffer insertion or structural bottlenecks in the system design.

Relating to Section 5, these information are used for quantitative evaluation of qualitative system properties.

3.1 Execution Time Models

As already mentioned in the introduction, best-case and worst-case execution times are not sufficient for qualitative design evaluation. For representation of qualitative timing properties of a CDG edge, we have to extend the basic CDG timing representation. Two different execution time models are introduced. For the calculation of the behavior concerning the real order of execution times, we extend it with a vector of ordered execution times $ls(e)$ with $e \in E_{CDG}$. This means that each particular execution time of each edge e is stored. This enables performance analysis of the system without any need for determining execution times again. We assume a deterministic temporal behavior of the components with regard to the same input data. We extend the basic definition of the CDG with

- The function $ls_{CDG}(e) : E_{CDG} \rightarrow \mathbb{N}_0^n$

$$\text{with } ls(e) = (ls_1, ..., ls_n) \tag{1}$$

For our probabilistic estimation approach, we assume that the edge e can have n different execution times $lp_1, ..., lp_n$. These execution times appear with probabilities of $P(lp_1), ..., P(lp_n)$. The execution times and probabilities can be calculated from profiled execution times represented by $ls(e)$. Therefore, the number of occurrence of each different execution time $ls_i \in ls(e)$ is summed up as n_i and divided by the number n of all profiled execution times. $lp_j = ls_i$ and $P(lp_j) = n_i/n$. Furthermore, $lp(e)$ can be taken from a specification or assumed by the designer.

We extend the basic definition of the CDG with

- The function $lp_{CDG}(e) : E_{CDG} \rightarrow (\mathbb{N}_0 \times \mathbb{R}^+)^n$

$$\text{with } lp(e) = \begin{pmatrix} lp_1, P(lp_1) \\ \vdots \\ lp_n, P(lp_n) \end{pmatrix}, n \in \mathbb{N} \text{ and} \tag{2}$$

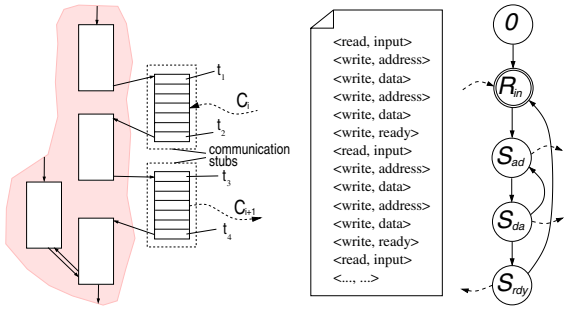
$$\sum_{i=1..n} P(lp_i) = 1 \tag{3}$$

3.2 Activity Model Elaboration

In this section, we briefly introduce our method for elaborating the CDG analysis model from execution traces of instrumented SystemC models. The ordered vector of communication endpoints CE that was derived from untimed simulation is used as input of the algorithm. The $ce \in CE$ are characterized by unique IDs. The algorithm assumes that a deterministic control-flow according to the order of communication endpoints exists. Nevertheless, the algorithm is able to incorporate branches in the control-flow even if the branches end at different communication endpoints. If the branch selection is changing indeterministically, the control-flow is unrolled. If the branch selection is deterministic, the algorithm detects recurring states and inserts backward edges in the elaborated process representation.

The input of the algorithm is the vector CE of communication endpoints. Each communication endpoint is characterized by its blocking property, and whether it is $send$ or $receive$. It returns a set of nodes V_{res} and a set of edges E_{res} as a representation of one CDG process. At the beginning, new nodes are inserted as long as no other node with the same ID exists in V_{cur}. When a node with an already existing ID is inserted, the assumption is that a backward edge to the existing node is inserted. The current state is stored. Then the next nodes from CE are compared with the existing successors of the already existing node in the CDG. If there is a deviation between the order of the nodes in CE and the already inserted nodes in V_{cur}, the algorithm uses backtracking and restores the state that was saved before inserting the backward edge. After that, a new node and an edge connecting the node with its successor are inserted. Otherwise, if the same node from which the backward edge was created is reached, the insertion of a backward edge was correct and the state of the system after inserting the nodes from CE is stored.

A small elaboration example is shown in Figure 4b. The list of communication endpoints depicted on the left side is elaborated to the process representation on the right side.



(a) Execution Time Determination (b) Elaboration Example

Figure 4: CDG Elaboration

Later, execution times are annotated to the edges of the elaborated graph using the ID of enclosing communication endpoints and the order of communication instances.

4. COMMUNICATION ANALYSIS

The objective of communication analysis is the determination of waiting times in blocking communication endpoints. The time a process has to wait in a blocking endnode of communication instance $C = (v_s, v_r)$ of a CDG is denoted by the slack variable $x(C)$. This time is calculated beginning at the last communication instance $C_{pre} = (v_n, v_b)$ that potentially synchronizes the two processes. With an iterative algorithm, we are using already determined slack values $x(C_i)$ for the calculation of subsequent ones. The algorithm terminates if the vector of system wide slack values recurs. Therefore, the following operators for path calculation have to be defined. Their definition for both models of computation as well as illustrative examples are given later.

1. Summing up latencies $\oplus$
2. Calculation of slacks in blocking communication nodes by subtraction of latencies $\ominus$
3. Incorporation of slacks in blocking communication nodes $\otimes_+$
4. Traversing a (potentially) non-synchronizing communication instance $\otimes_-$
5. Starting path calculation from (potentially) unsynchronized communication $|x(C)|_-$

The condition for synchronization and the quantitative determination of waiting times can be formulated by the following two equations:

$$x(C) = (|x(C_{pre})|_- \oplus l(v_b \rightsquigarrow v_s)) \ominus (l(v_n \rightsquigarrow v_r) \quad (4)$$

$$x(C) = l(v_n \rightsquigarrow v_s) \ominus (|x(C_{pre})|_- \oplus l(v_b \rightsquigarrow v_r)) \quad (5)$$

The selection of one of these equations depends on the position of the blocking node v_b in the control-flow of the processes, whether it is prior to v_s or prior to v_r.

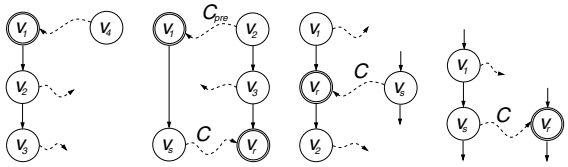
The equations mean that the control-flow of a process has to wait if it arrives at the blocking communication endpoint of a communication instance before the control-flow of the communication partner arrives at the non-blocking endpoint. The application of the operators has to be defined for the two presented models.

4.1 Execution Time Vectors

The vector $ls(e)$ can be used for determining the same performance characteristics as determined by a simulation of the system of communicating processes. A modification of one component only demands for determining the execution time characteristics of this component as long as the functionality does not change. In comparison to our probabilistic estimation approach (Section 4.2), we need to store the ordered vector of execution times $ls(e)$ instead of summing up the occurrence of equal execution times. Depending on the environment of each component, these data can grow up to some gigabytes for each component configuration. For the i-th utilization of an edge e_{CDG} in system analysis, the i-th value of $ls(e)$ is used. The operations $\oplus$ and $\ominus$ are treated as $+$ and $-$. The incorporation of a slack in a blocking communication node by the operation $\oplus_s$ is handled as $+max(0, x(C))$. Traversing a potentially unsynchronized communication instance is treated as $+max(0, -x(C))$. The operation $|x(C)|_-$ is handled as $max(0, -x(C))$. Although we are using max expressions in this paper, we do not define a new semantics for the $(max, +)$ algebra.

4.2 Execution Time Distributions

Here we describe our proposed analytic approach for probabilistic estimation of the system behavior. Our model of computation assumes that each execution time of edge e_1 can be followed by each execution time of edge e_2. Obviously, this blind combination ignores potential dependencies of code blocks and may lead to a misprediction of system properties itsself. As presented in Section 2, several approaches exist to identify dependent code blocks for WCET [5, 10] and worst-case performance analysis [21]. Experimental results in Section 6 show that the deviation between the probabilistic estimation and the real behavior is under 15%. Nevertheless, we work on an extension to the proposed method by applying an additional correction operator for incorporating depencies of execution times. This method can be integrated in the proposed design-flow but this issue is out of scope of this paper. lp is applied instead of l for denoting latency distributions. Although the needed operations are based on basic probability theory, issues according the incorporation of waiting times and non-synchronizing communication have to be considered. Waiting times at communication endpoints can not be simply combined with each different execution time of the path leading to this CDG node. When waiting times at blocking CDG nodes are incorporated in path calculation, it has to be ensured that they are only combined with the execution times they were calculated from. Otherwise, the analysis itsself would lead to a misprediction of the system behavior. Furthermore, if



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure 5: Operand Application Examples

communication is not or not always synchronizing the control-flow of the processes due to timing issues this impact has to be incorporated as well. This needs to be defined with regard to a traversion of a non-synchronizing communication instance and the incorporation of waiting times at the blocking communication endpoint. The blocking property of a node is illustrated by two concentric circles surrounding it. Below, the five mentioned operators are defined and illustrated by the examples in Figure 5.

(1.) Concerning the example depicted in Figure 5a that considers two subsequent edges (v_1, v_2) and $(v_2, v_3) \in E_{CDG}$, with $a = lp(v_1, v_2), b = lp(v_2, v_3), \tau(v_2) = \{send_{async}, receive_{async}\}$, the latency $lp(v_1 \rightsquigarrow v_3)$ is denoted by:

$$lp(v_1 \rightsquigarrow v_3) = a^{(m)} \oplus b^{(n)} = c^{(m \cdot n)}$$
$$c_{(i-1)n+j} = (a_i + b_j, P(a_i) \cdot P(b_j)) : 1 \leq i \leq m, 1 \leq j \leq n$$

(2.) For calculating the time the control-flow has to wait at communication instance $C = (v_s, v_r)$ depending on the paths $lp(v_1 \rightsquigarrow v_s) = a$ and $lp(v_2, v_r) = b$ from a previous communication instance $C_{pre} = (v_2, v_1)$ that synchronizes the communicating processes, the result of

$$\begin{aligned} x(C) &= a^{(m)} \ominus b^{(n)} = c^{(m \cdot n)} \\ c_{(i-1)n+j} &= (a_i - b_j, P(a_i) \cdot P(b_j)) : 1 \leq i \leq m, 1 \leq j \leq n \end{aligned}$$

consists of tupels of time x and probability $P(x)$, the control-flow has to wait in the blocking communication endpoint. Figure 5b presents a simple example.

(3.) Like illustrated in Figure 5c two edges (v_1, v_r) and $(v_r, v_2) \in E_{CDG}$ with $a = lp(v_1, v_r)$ and $b = lp(v_r, v_2)$ and a communication instance $C = (v_s, v_r)$ with $\tau(v_r) = receive_{sync}$ are given. For incorporating the time the control-flow waits in v_r we define:

$$\begin{aligned} lp(v_1 \rightsquigarrow v_2) &= (a^{(m)} \otimes_+ x(C)^{(p \cdot m)}) \oplus b^{(n)} = c^{(p \cdot m)} \oplus b^{(n)} \\ c_{(i-1)p+q} &= (a_i + max(-x(C)_{(i-1)p+q}, 0), P(x(C)_{(i-1)p+q})) \\ & \quad 1 \leq i \leq m, 1 \leq q \leq p \end{aligned}$$

(4.) Given are a communication instance $C = (v_s, v_r)$ and a path $(v_1 \rightsquigarrow v_s)$ with $lp(v_1 \rightsquigarrow v_s) = a$. $\tau(v_s) = send_{async}, \tau(v_r) = receive_{sync}$. Then

$$\begin{aligned} lp(v_1 \rightsquigarrow v_r) &= a^{(m)} \otimes_- x(C)^{(p \cdot m)} = c^{(p \cdot m)} \\ c_{(i-1)p+q} &= (a_i + max(-x(C)_{(i-1)p+q}, 0), P(x(C)_{(i-1)p+q})) \\ & \quad 1 \leq i \leq m, 1 \leq q \leq p \end{aligned}$$

incorporates the timing impact of traversing a potentially unsynchronizing communication instance C. An example scenario is depicted in Figure 5d.
(**5.**) When the synchronization condition in Equation 4 and 5 for a communication instance C is formulated, the influence on timing effected by a potentially non-synchronizing previous communication instance C_{pre} has to be considered. The operation $|x(C_{pre})|_-$ is defined by:

$$\begin{aligned} |x(C)^{(k)}|_- &= c^{(k)} \\ c_l &= (max(x(C)_l, 0), P(x(C)_l)) \\ & \quad 1 \leq l \leq k \end{aligned}$$

It is not necessary to calculate the entries in the slack matrix if no negative entry will be contained in it. In that case, these values are not needed and can be removed for optimization issues. The term $a \otimes_+ x(C)$ can be used instead for symbolic slack incorporation. As an example, $lp(v_1, v_s) = a$, $lp(v_2, v_r) = b$, $lp(v_r, v_3) = c$ and the asynchronous communication instance $C = (v_s, v_r)$ are given. The previous synchronizing communication instance was (v_1, v_2). C is completely synchronizing. The calculation of $lp(v_2 \rightsquigarrow v_3)$ is defined by:

$$\begin{aligned} lp(v_2 \rightsquigarrow v_3) &= (b^{(n)} \otimes_+ x(C)^{(m \cdot n)}) \oplus c^{(k)} \\ &= (b^{(n)} \otimes_+ (a^{(m)} \ominus b^{(n)})) \oplus c^{(k)} \\ &= (b_j + a_i - b_j, P(a_i) \cdot P(b_j)) \oplus c^{(k)} \\ &\quad \text{Due to condition (3)} \\ &= (a_i, P(a_i)) \oplus c^{(k)} = a^{(m)} \oplus c^{(k)} \\ &\quad 1 \leq i \leq m, 1 \leq j \leq n \end{aligned}$$

The application of communication analysis is briefly presented in Section 5.

4.3 Classification of Execution Times

For reducing computational effort of the probabilistic estimation approach, different execution times can be classified to groups. Although this may lead to a loss of accuracy, it provides an effective method for deriving an initial overview about the qualitative system behavior with lower computational effort. Arbitrary classification algorithms can be applied. Examples are a division of the execution time distribution of an edge to equidistant intervals or a classification depending on a maximal distance ϵ_t to its neighbors. All execution times in each interval are combined to one execution time and weighted depending on their probabilities. The probabilities are summed up. If each interval has a size of n, the maximum number of operations m (for e.g. $\oplus$) is reduced to m/n^2.

5. QUANTITATIVE EVALUATION

In this section, we briefly want to present the application of our novel approach for qualitative system analysis that is based on an approach for quantitative system analysis [18]. Our approach for performance analysis uses the information on waiting times in blocking communication endnodes that are determined using communication analysis. One direct result from communication analysis is the identification of permanently non-synchronizing communication instances. This leads to data loss due to high input rates. The risk of data loss can be determined using our methodology as well as the impact of buffer insertion. These information can be used for detecting bottlenecks and for explorating the underlying system architecture. The same operators as defined in Section 4 are used for determination of the performance characteristics. We are not presenting new analysis methods for system properties in this paper. We describe the quantitative evaluation of qualitative properties between best-case and worst-case. Our extension to the analysis methodology can be applied to previously published analysis methods, like e.g. the determination of worst-case response time (WCRT), system latency, component utilization or access conflicts on shared communication resources [19] for performance risk evaluation. In Section 6, we use the end-to-end system latency between the input v_I and the output v_O that can be determined by computing $l(v_I \rightsquigarrow v_O)$ as example.

The analysis does not only allow the determination of computation resource characteristics. The information on communication instances and the temporal relation between them can be applied to determine the impact of the communication infrastructure on the system characteristics. Therefore, typical parameters like latency and transmission time of communication instances are incorporated. The proposed approach allows an exploration towards the functionality, the environment, the mapping, and the underlying platform. A modification of components or an exchange of complete subsystems can be reflected without completely analyzing the entire system again.

6. EXPERIMENTAL RESULTS

In this section, we present experimental results from the application of our methodology. According to the flow presented in Section 3, we started with a SystemC model of a Viterbi decoder consisting of nine processes and mapped the two processes with the main functionality, `vit_fwd` and `vit_bwd` each to a PowerPC 604 with 100 MHz. The resulting elaborated CDG representing the inter-process communication structure is depicted in Figure 6.

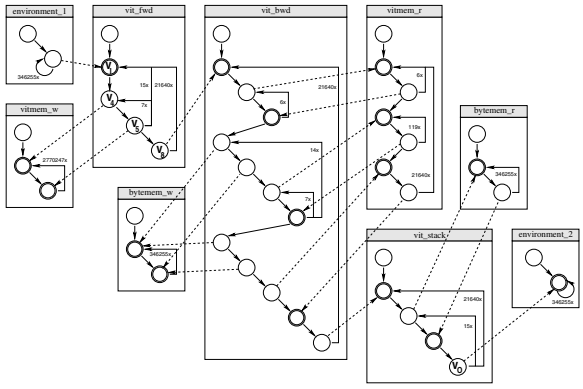
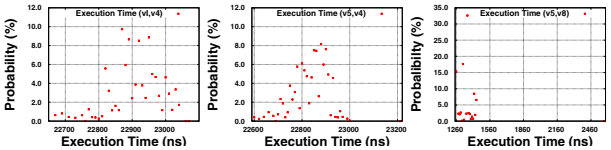


Figure 6: Viterbi Decoder CDG

The edge weights characterizing the latencies of the four memories were annotated with 60 ns from the memory specification. The same latencies were used for the module `vit_stack` that has the same latency characteristics like memory. We profiled the execution times relating to two different environments the decoder processed. We used a 1 MB *pgm* image as one input and a 320 KB *XML* file as another. The data packages arrived with an inter-arrival time of 160.3 ms. Of course, this arrival time model can be modified using probabilities, periods and jitters. Some execution time distributions of Viterbi CDG edges determined using the *pgm* image are depicted in Figure 7. It is apparent that these distributions do not correspond to common statistical distributions often used in theoretical models. Then we used the information derived from



(a) $lp(v_I \rightsquigarrow v_4)$ (b) $lp(v_5 \rightsquigarrow v_4)$ (c) $lp(v_5 \rightsquigarrow v_8)$

Figure 7: Execution Time Distributions

profiling and calculated the end-to-end latency $lp(v_I \rightsquigarrow v_O)$ of the entire design between the nodes v_I and v_O using both approaches presented in Section 4 and Section 5. These results are depicted in Figure 8. With a growing amount of input data, the upper and lower bounds of the packet latency calculation using execution time vectors (I.) were getting closer to the limits of the probabilistic estimation approach(II.). Please note that the intervals determined using static timing analysis [18] are much larger, but they can be also included to the execution time distribution. A closer inspection

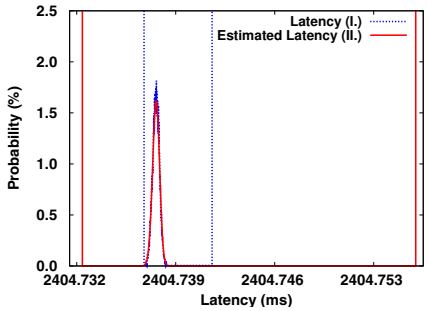
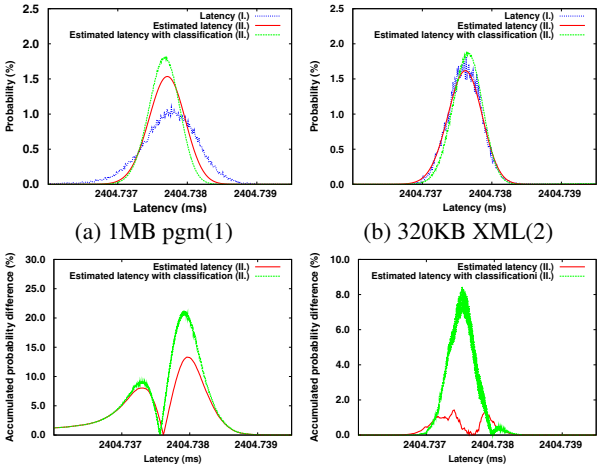


Figure 8: Viterbi Decoder Packet Latency

on a comparison between packet latency calculation using execution time vectors, probabilistic latency estimation and probabilistic latency estimation with run-time classification is depicted in Figure 9a and 9b. The classification uses equidistant intervals of 50 ns. For a comparison of the three results, we subtracted the accumulated values of both estimated curves from the curve using execution time vectors. These results are depicted in Figure 9c and 9d. The maximum deviation of the estimated curve without classification is below 15% using the *pgm* and below 2% using the *XML* as input. This shows moreover that the used input data have an impact on the typical design latency. The application of classification leads to a maximum deviation of the estimated curves below 25% using the *pgm* and below 10% using the *XML*. The applied classification algorithm reduced the number of necessary arithmetic operations by 35%. Concering the *XML* input data, the probabilistic estimation without classification needs 34% less arithmetic operations than using execution times vectors. The related execution time profiles ls need 1.3 GB of storage.



(a) 1MB pgm(1) (b) 320KB XML(2)

(c) Accumulated Difference(1) (d) Accumulated Difference(2)

Figure 9: Comparison of Accuracy

7. CONCLUSION AND FURTHER WORK

The early availability of quantitative design characteristics for comparing system requirements with design properties is valuable for designers. Unlike other approaches in this area, we did not focus on the determination of pessimistic best-case and worst-case quantities of system properties. Our proposed analysis methodology determines qualitative numbers between best-case and worst-case of system properties and quantifies them with probabilities. Relating this issue, we developed a hybrid approach for incorporating execution time profiling and system-level analysis techniques. We presented two different models of computation: One for incorporating execution time vectors from profiling and one that combines all execution times to a distribution function. The determined quantities can be used to evaluate the performance risk of a design relating to requirements from the specification. We implemented our analysis methods and presented experimental results from a real-world design, a Viterbi decoder. The comparison of both approaches shows, the hybrid probabilistic estimation approach allows quite accurate requirement evaluation at system-level. Next steps towards an enhancements of the probabilistic estimation accuracy are the application of techniques known from WCET analysis for incorporating dependencies between code fragments at this high level of abstraction.

8. REFERENCES

[1] K. Albers, F. Bodmann, and F. Slomka. Hierarchical event streams and event dependency graphs: A new computational model for embedded real-time systems. In *ECRTS '06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 97–106, Washington, DC, USA, 2006. IEEE Computer Society.

[2] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf scheduling. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 492–497, Washington, DC, USA, 2005. IEEE Computer Society.

[3] R. Alur. Timed Automata. In *Proceedings of Computer-Aided Verification*, 1999.

[4] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[5] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time systems. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, page 279, Washington, DC, USA, 2002. IEEE Computer Society.

[6] S. Bradley, W. Henderson, and D. Kendall. Using Timed Automata for Response Time Analysis of Distributed Real-Time Systems. In *Proceedings of Workshop on Real-Time Programming WRTP*, 1999.

[7] J. Bryans, H. Bowman, and J. Derrick. Model checking stochastic automata. *ACM Trans. Comput. Logic*, 4(4):452–492, 2003.

[8] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proceedings of DATE*, Munich, 2003.

[9] N. Dhanwada, R. A. Bergamaschi, W. E. Dungan, I. Nair, P. Gramann, W. E. Dougherty, and I.-C. Lin. Transaction-level modeling for architectural and power analysis of PowerPC and CoreConnect-based systems. *Springer: Design Automation for Embedded Systems*, 2006.

[10] S. V. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal. Automatic scenario detection for improved wcet estimation. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 101–104, New York, NY, USA, 2005. ACM Press.

[11] M. Krause, O. Bringmann, and W. Rosenstiel. Target Software Generation: An Approach for Automatic Mapping of SystemC Specifications onto Real-Time Operating Systems. *Springer: Design Automation for Embedded Systems*, 2007.

[12] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of applications with stochastic task execution times. *Trans. on Embedded Computing Sys.*, 3(4):706–735, 2004.

[13] R. Marculescu and A. Nandi. Probabilistic application modeling for system-level performance analysis. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 572–579, Piscataway, NJ, USA, 2001. IEEE Press.

[14] M. A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, 1984.

[15] W. Müller, W. Rosenstiel, and J. Ruf, editors. *SystemC: methodologies and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[16] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and optimization of distributed real-time embedded systems. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 593–625, New York, NY, USA, 2004. ACM Press.

[17] S. Schliecker, M. Ivers, and R. Ernst. Integrated Analysis of Communicating Tasks in MPSoCs. In *CODES+ISSS '06*. ACM Press, 2006.

[18] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Worst-case performance analysis of parallel, communicating software processes. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, 2002.

[19] A. Siebenborn, A. Viehl, O. Bringmann, and W. Rosenstiel. Control-Flow Aware Communication and Conflict Analysis of Parallel Processes. In *Proceedings of the 12th Asia and South Pacific Design Automation Conference ASP-DAC 2007, Yokohama, Japan*, 2007.

[20] W. Stark and S. A. Smolka. Compositional Analysis of Expected Delays in Network of Probalistic I/O Automata. In *IEEE Symposium on Logic in Computer Science*, 1998.

[21] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 450–459, 2004.

[22] T. Wild, A. Herkersdorf, and G.-Y. Lee. TAPES–Trace-based architecture performance evaluation with SystemC. *Springer: Design Automation for Embedded Systems*, 2006.

[23] Y. Xie and W. Wolf. Allocation and Scheduling of Conditional Task Graph in Co-Synthesis. In *Proceedings of DATE*, Munich, 2001.

[24] A. Yakovlev, L. Gomes, and L. Lavagno. *Hardware Design and Petri Nets*. Kluwer, 2000.

[25] T.-Y. Yen and W. Wolf. Performance Estimation for Real-Time Distributed Embedded Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 9, November 1998.