Journal of Theoretical and Applied Information Technology, Islamabad PAKISTAN

31st May 2010. Vol.15. No.2.
© 2005-2010 JATIT. All rights reserved

www.jatit.org

A METHODOLOGY FOR ESTIMATION OF PERFORMANCE IN REAL TIME DISTRIBUTED EMBEDDED SYSTEMS

[1]RAMESH BABU NIMMATOORI, [2]Dr. VINAY BABU A, [3]SRILATHA C

[1] Assoc Prof., Department of CSE, ASTRA, Hyderabad, India-500008
[2]Professor, Department of CSE, JNTUH, Hyderabad, India-500085
[3] Assoc Prof., Department of ECE, ASTRA, Hyderabad, India-500008

ABSTRACT

As embedded systems become increasingly networked, and interact with the physical world, Real Time Distributed Embedded Systems are emerging. This research work mainly focuses on the performance estimation and analysis of Real Time Distributed embedded systems. Performance evaluation is one of the key challenge in the system analysis. Major design parameters that influence the performance includes real-time properties, such as ask execution times, communication delays, the degree of parallelism in computations, and the throughput of the communication architecture. Here the performance estimation is done by discrete event simulation methodology which is based on repetitive simulations of the model, and therefore does not suffer from memory consumption limits. Moreover, it can provide partial results in case the models are too large for exhaustive analysis. They provide the platform for the implementation of cyber-physical systems that run in open environments, in less predictable conditions by providing a highly adaptive infrastructure for reusable resource management services than previous generations of real-time embedded systems that are specialized for specific application domains. The proposed algorithm is scalable for large-scale systems as well, and performs well for finding task mappings that satisfy real-time constraints. Obtaining a feasible mapping for a model of this size will be in the order of seconds. The proposed algorithm is implemented and the results are compared with that of Random Simulations.

Keywords: *Embedded Systems, Real Time, Distributed Systems, Performance, Task, Event.*

1. INTRODUCTION

Distributed system is one where data are located on several computers that are linked together by a heterogeneous network. The advantages of such system are increased availability of resources, increased reliability, and increased execution speed in less time. The coordination of activities among computers is a complex task. If a transaction runs across two sites, it may commit at one site and may failure at another site, leading to an inconsistent transaction.

A distributed real-time system is an integrated system comprising a set of dedicated hardware that monitors real-world phenomena, acts and reacts on events within specified time period. A real-time system is application driven: its requirements are dictated by the outside environment not by the computer. The rapidly expanding application area poses a constant pressure to the computing community to improve skills and techniques used for the design and development of such systems. Real-time technology has always been considered as one of the most complex areas of computer science. Time-constraint information processing is not only about efficiency, it involves timely response, correctness, concurrency, distribution, modelling, assessment and analysis.

In several domains, such as military, industrial automation, and financial markets, message latency plays a critical role. Since it is technically hard and in most cases costly to guarantee that each and every message is delivered within a predefined period of time (hard-real time), many applications impose weaker requirements by allowing a small portion of messages to exceed their deadline (soft real-time). Still, as a consequence of application complexity and the volatile nature of the distributed environment, even compliance with these weaker constraints is a challenging task. Unexpected activity bursts, message loss due to unreliable communication medium, network buffer overflow, network congestion, resource sharing and many other unpredictable factors may result in significant increase in the end-to-end message delay. While, in the automotive domain, the modern active-safety applications consist of complex end to-end computations that collect data from 360^o sensors around the vehicle to understand the positioning of surrounding objects and detect hazardous conditions. On hazard detection, active safety functions attempt to inform the driver or provide control overlays to reduce the risk. Most of these functions are high-level controls which drive low-level actuation loops, but they are nevertheless subject to timing constraints. Such active-safety applications are typically run on distributed architectures. Distributed architectures supporting the execution of hard real-time applications are common not only for automotive, but also for avionics and industrial control systems. As embedded systems become increasingly networked, and interact with the physical world, Real-time Distributed Embedded systems emerge. They systems range from small-scale Multi-processor Systems-on-Chip operating in resource constrained environments such as cell phone platforms, medical devices and sensor networks all the way to large-scale software-intensive systems of systems used in avionics, ship computing environments, and in supervisory control and data acquisition systems managing regional power grids.

2. RELATED WORK

A generic, component-based formal framework for the scheduling analysis and formal performance evaluation of platform-based embedded systems was proposed in [1]. The authors utilize event streams to model communication characteristics of tasks. SymTA/S [2] is a formal analysis tool that applies methods from scheduling theory and symbolic simulations for the performance analysis of complex heterogeneous Multi-processor System-on-Chip. This approach can provide bounds on end-to-end latencies, bus and processor utilization, and worst-case scheduling scenarios. Modular Performance Analysis [3] is an approach to characterize Distributed Embedded systems merely by describing incoming and outgoing event rates, message sizes, and execution times. Resources and the distributed execution platform is defined in similar terms, and Real-Time Calculus is then used to compute upper and lower bounds of the system performance.

Although all static analysis methods provide scalable solutions for performance evaluation they cannot model dynamic effects, such as varying delays and race conditions, as they do not capture the flow of data, and are less accurate than dynamic estimation methods. Communication in embedded systems is often non-deterministic, data-dependent, and hard to model as well-formed event streams.

In contrast, the analysis methods in this research work explicitly capture dependencies, and the asynchronous event-based communication of asynchronous event-driven Distributed Real-time Embedded systems. This approach captures dynamic effects, such as varying delays and race conditions in distributed systems, and results in more accurate performance analysis at the price of being computationally more intensive.

Simulations are the preferred and widely accepted way to evaluate the performance of Distributed Real-time Embedded system designs in the industry today. A simulation based design space exploration method, however, has several disadvantages. Developing the models for a design alternative may take weeks or months therefore only a handful of alternatives may be practically analyzed given the short product development cycles. Moreover, designers typically notice performance issues late in the design cycle therefore addressing changes can be rather time-consuming and costly. Register-Transfer Level (RTL) languages such as VHDL [4] and Verilog [5] are classic hardware description languages that target hardware specification at low-level abstractions providing a high precision, synthesizable platform for hardware development. The low-level abstraction, however, results in slow simulation speeds unsuitable for the analysis of complex Multi-processor System-on-Chip. Due to the increase in Multi-processor System-on-Chip design complexity as well as the decrease in the time to market window, today's designers are turning to transaction-level modelling languages such as SystemC [6] and SystemVerilog [7] to perform early design exploration and hardware-software co-design in order to shorten the design cycle. Transaction-level modeling focuses on the interactions between systems components, such as bus transfers, interrupts or signals, rather than on gates or registers. Transaction-level languages employ higher-level abstractions than RTL languages and are often not synthesizable. A semiformal simulation-based performance evaluation method for Multi-processor System-on-Chips was proposed in [8]. The authors represent execution traces as symbolic graphs for performance analysis, annotated with execution times obtained by simulating individual components of the system. Although the approaches described in [8] improves simulation speed by utilizing symbolic representations of execution traces, the quality of results depends on the ad-hoc selection of test vectors.

This research work considers the problem of combining simulations and formal methods for real-time analysis. Simulations in our approach are used to obtain execution intervals that we use to annotate the formal models for design space exploration. The symbolic model introduced captures all possible execution traces of the system, not just one execution trace. This is a more accurate model for Real Time Distributed Embedded systems, where execution times are rarely constant. Moreover, the proposed methodology is formalized for obtaining test vectors based on the discrete event model, that provide better coverage than random simulations.

A generic method for protocol verification using synchronous protocol automata is presented in [9]. Synchronous protocol automata can be mapped to the Finite State Machine (FSM) Model of Computation, and a main contribution of the paper is to show how protocols can be translated to a formal language for functional verification. A method for the functional verification of the Peripheral Component Interconnect (PCI) protocol is described in [10]. The authors model the PCI protocol by the FSM Model of Computation, and use the Cadence SMV tool for functional verification. A similar approach is used to verify the IBM Core Connect arbiter in [11]. An early work on applying model checking methods to the ARM Advanced Microcontroller Bus Architecture Advanced High-speed Bus protocol was presented in [12], where the authors used FSM models and the SMV tool to uncover an unspecified condition. The described case study is due to awed implementation rather than the protocol itself. A verification platform for Advanced Microcontroller Bus Architecture ARM7 is presented in [13]. The authors use the SMV tool to prove the functional correctness of the protocol by checking various properties. The authors do not describe any ambiguities, rather they focus on properties that have turned out to be valid. A verification platform for Advanced Microcontroller Bus Architecture Advanced High-speed Bus protocol using a combination of model checking and theorem proving is described in [14]. The author extends earlier approaches by considering both control and data properties, and describes properties that have proven to be true.

3. APPROACH

Discrete event based simulation methodology is proposed for the performance evaluation of Real Time Distributed embedded systems. This methodology employs fixed priority scheduling. The concept of logical executing time and the event order tree are introduced in this methodology. The proposed methodology explicitly captures the flow of data and communication effects (such as non-deterministic delays etc.). This approach represents real-time properties in continuous time leading to a minimal memory requirements.

The main advantage of the proposed methodology is that it gradually increases coverage over time. Random simulation-based methods do not have this property. Random simulations assign execution times following a uniform distribution from the intervals of tasks. However, in the actual system the execution times rarely follow a uniform distribution; it is quite probable that some execution times are more frequent than others, and that the real system encounters execution traces that were not considered during the simulation-based evaluation process. Since these traces are not simulated, designers will also fail to recognize how the system performance/ schedulability might change due to dynamic effects such as race conditions. Therefore, we conclude that random simulations may be useful for the first steps of performance evaluation. The preliminary results show that the proposed methodology allows better simulation performance compared to the actual simulations with comparable accuracy, providing an efficient method for design space exploration.

Discrete event based simulation:

This involves the Analysis Language for the formal performance analysis of Real-time Distributed Embedded systems. This method is applicable to non-preemptive systems only, but can be composed with the real-time verification of preemptive systems. We introduce a formal model

31st May 2010. Vol.15. No.2.
© 2005-2010 JATIT. All rights reserved
www.jatit.org

based on discrete event scheduling using the concept of logical execution time, and the Event Order Tree. Nodes in the EOT represent events, and edges represent causality between the events. As events may arise non-deterministically, the tree may branch when different event orderings are possible. The proposed model explicitly captures the flow of data and communication effects (such as non-deterministic delays etc.) in event-driven systems for dynamic performance evaluation.

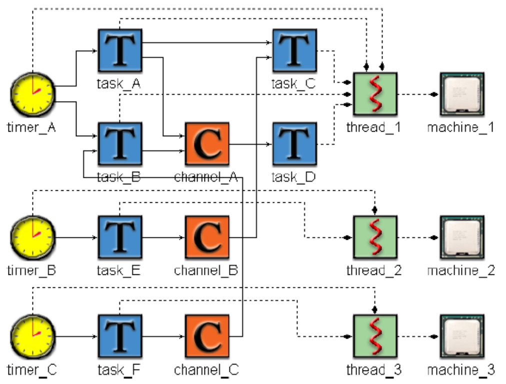

Figure 1: Example model of a Real-time Distributed Embedded system

Figure 1 shows an example of Real-time Distributed Embedded systems. Timers are represented by the timer icon, tasks by the T icon, channels by C, threads by the thread icon, and machines by the processor icon. The solid arrows show the dependencies (set D) in the model. The mapping of tasks and timers to threads (the function *thread*(t_k) : T →TH) and the mapping of threads to machines (the function *machine*(t_k) : TH→M) are shown by the dashed arrows. Channels are not mapped to any threads as they are executed non concurrently. Since there is only one thread assigned to each processor, the model utilizes non-preemptive scheduling.

The following notation is employed:

The sequence of input events of a task t_k as:

$I_k = \{ i_{k0,} i_{k1,\ldots\ldots} \}$

The sequence of output events of a task t_k as:

$O_k = \{ o_{k0,} o_{k1,\ldots\ldots} \}$,

where. t € T,

$i_{k0,} i_{k1,\ldots}$ € E,

$o_{k0,} o_{k1,\ldots}$ € E.

E is the set of all (infinite) events generated by the system.

End-to-end computation time problem formulation:

We define the end-to-end computation time between an input event i_{jn} of task t_j and an output event o_{km} of task t_k as the maximum possible difference between the events' timestamps along all the possible runs of the model.

$end\text{-}to\text{-}end(o_{km}, i_{jn}) = max[time(o_{km})\text{-}time(i_{jn})]$

If task t_k does not depend on task t_j then $end\text{-}to\text{-}end(o_{km}, i_{jn} = \infty$

It is a fact that the product of local worst case execution times does not necessarily result in worst case end to-end computation times.

The following is the algorithm developed based on Discrete events:

```
Step 1. Run simulation during which each task
        stores its start time as $start_k$

Step 2: During the simulation all tasks $t_k$ observe
        events $e_i$ that are raised in the [$start_k$
        + $bcet_k$, $start_k$ + $exec\_time_k$] interval

Step 3: **If** startk + next timek < time(ei) then
            // a branching point encountered
        in the (bcetk, wcetk) interval//
            store the value of time($e_i$) - $start_k$
        in the next $time_k$ variable
        **else**
        do nothing, event will be considered in
        subsequent simulations
        **end if**

step 4: // find all race conditions with the current
        $exec\_time_k$ assignments//

        **for** all race conditions detected between
        events $e_i$; $e_j$,… $e_k$ during the simulation
        do
        search for the set containing events $e_i$;
        $e_j$,… $e_k$ in superset R
        **if** the set is found then
        do nothing
        **else**
        add the set S = {$e_i$; $e_j$,… $e_k$} to the superset
        R
        **end if**
        **end for**
```

31st May 2010. Vol.15. No.2.
© 2005-2010 JATIT. All rights reserved
www.jatit.org

4. RESULTS

We now demonstrate this problem in non-preemptive systems to motivate our approach for formal performance evaluation using the simple example shown in Figure1. We define the period of each timer to be 100 time units, and the delay of each channel to be 0. Figure 2 illustrates the first period of model execution traces shown in Figure 1 using the parameters in Table 1. In this example, for most tasks the Best Case Execution Time $bcet_k$ time equals the Worst Case Execution Time $wcet_k$ time to reduce complexity, for easier illustration. We utilize fixed-priority scheduling in machine 1 between tasks t_A, t_B, t_C and t_D. Tasks t_E and t_F are executed concurrently and have their own schedulers. Note that Earliest Deadline First (EDF) scheduling would result in a deadline miss by task t_B, as it scheduled the sequence t_A, t_C, t_B.

This illustrates that EDF is not optimal in the non-preemptive systems. The execution trace in the left of Figure 2 demonstrates that the system is schedulable if all tasks execute using their Worst Case Execution Time (WCET). The trace in the middle of Figure 2 shows that the system is schedulable when Best Case Execution Time (BCET) are considered during the execution trace. However, the trace in the right of Figure 2 shows that task t_C might miss its deadline if task t_E executes for 71 time units. This example shows that the performance evaluation of event driven non-preemptive DRE systems has to consider execution intervals rather than worst case execution times, and justifies the need for formal performance analysis.

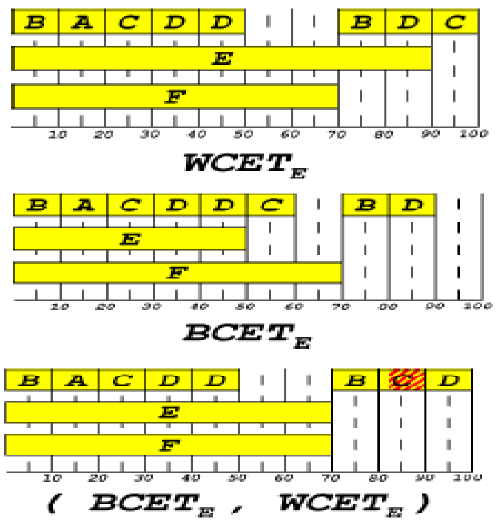

Figure 2: execution of traces of the example shown.

Task	t_A	t_B	t_c	t_D	t_E	t_F
bcet	10	10	10	10	50	70
Wcet	10	10	10	10	90	70
deadline	22	25	12	32	100	100

Table 1: timing details for the model shown.

In a Real-time Distributed Embedded systems semantic domain, tasks receive a potentially infinite sequence of events (timestamped values as event labels) in chronological order. The task then outputs a timestamped event for each input event. The order of events in the output sequence of each task is the same as the order of events in the input sequence of the task. The timestamps of input events must be smaller than or equal to their corresponding output events. Note that the discrete event simulation is completely deterministic if timestamps are unique constants.

Comparison with Random Simulations:

The main advantage of the Discrete event based simulation method is that it gradually increases coverage over time. Random simulation-based methods do not have this property. Random simulations assign execution times following a uniform distribution from the $[bcet_k, wcet_k]$ intervals of tasks.

Let's denote the two endpoints of a branching interval as l_{ki}, h_{ki} , where l_{ki} refers to the lower bound on the branching interval, and h_{ki} refers to the higher bound on the branching interval and $bcet_k < l_{ki} < h_{ki} < wcet_k$. Then the probability that the random execution time is within the branching interval can be formalized as follows:

$$P = \frac{h_{k_i} - l_{k_i}}{\mathsf{wcet}_k - \mathsf{bcet}_k}$$

$$\text{Note that } \lim_{(h_{k_i} - l_{k_i}) \to 0} P = 0,$$

Therefore the probability that an exact number is chosen randomly from a continuous-time interval is close to 0, even if infinite number of simulations are executed. Also, the smaller the branching interval, the less chance that we actually consider it during simulation. Since there is a higher chance that the execution time is picked from larger branching intervals, repetitive simulations will pick execution times from branching intervals that have already been chosen for simulation. However, in the actual system the execution times rarely follow a uniform distribution; it is quite probable that some execution times are more frequent than others, and that the real system encounters execution traces that were not considered during the simulation-based evaluation process. Since these traces are not simulated, designers will also fail to recognize how the system performance/ schedulability might change due to dynamic effects such as race conditions or congestions. Therefore, it can be conclude that random simulations may be useful for the first steps of performance evaluation, but can achieve only partial coverage of the possible execution traces over time.

In contrast, the method presented gradually increases coverage over time. Moreover, each branching interval is considered only once, and the worst case times are checked directly, rather than a random number from the branching interval. Therefore, the proposed method can discover significantly more corner cases than random simulation-based performance estimation techniques.

TOOLS EMPLOYED

To check whether the observations are relevant in large-scale systems, the experiments to compare random simulations and the Discrete event based simulation method are run on an Intel Core i7 920 processor running at 4GHz using 6GB of three-channel RAM. On this test configuration, an open source analysis tool, Dream 0.7 Beta version is used for simulation.

5. CONCLUSIONS

The proposed method explicitly captures the data flow, and models the communication and execution intervals using a non-preemptive scheduling model. This leads to a formal executable model allowing to bridge the gap between simulations and formal verification. Our benchmarks based on a real time model case study show that the methodology employed for performance evaluation can achieve better coverage than alternative methods, and provides a way for the systematic measurement of coverage. The proposed approach allows to efficiently exploring large design spaces early in the design flow, provides formal guarantees on real-time constraints, and can produce counter-examples when real-time properties are violated.

REFERENCES:

[1] K. Richter, M. Jersak, and R. Ernst, “A Formal Approach to MpSoC Performance Verification”, IEEE *Computer,* 36:60-67, April 2003.

[2] R Henia and Arne Hamann and Marek Jersak and Razvan Racu and Kai Richter and Rolf Ernst, “System Level Performance Analysis - the SymTA/S Approach”. *IEEE Proceedings on Computers and Digital Techniques*, 152:148-166, 2005.

[3] Ernesto Wandeler and Lothar Thiele and Marcel Verhoef and Paul Lieverse, “System architecture evaluation using modular performance analysis - a case study”, *Software Tools for Technology Transfer (STTT*), 8(6):649-667, Oct. 2006.

[4] IEEE. VHDL (IEEE 1076 Standard), 2000.

[5] IEEE. Verilog *(IEEE 1364 Standard*), 2001.

[6] OSCI. SystemC ver 2.1 (*IEEE 1666 Standard)*, 2005.

[7] IEEE. SystemVerilog (*IEEE 1800 Standard*), 2005.

[8] 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,* 20:768-783, 2001.

[9] V. D'silva, S. Ramesh, and A. Sowmya, “Synchronous protocol automata: a framework for modelling and verification of SoC communication architectures”, *In IEEE Proceedings of Computers and Digital Techniques*, volume 152, pages 20-27, January 2005.

[10] P. Chauhan, E. M. Clarke, Y. Lu, and D. Wang, “Verifying IP-Core based System-On-Chip Designs” *In Proceedings of IEEE ASIC SOC Conference,* pages 27 - 31, 1999.

31st May 2010. Vol.15. No.2.
© 2005-2010 JATIT. All rights reserved
www.jatit.org

[11] A. Goel and W. R. Lee, "Formal Verification of an IBM CoreConnect Processor Local Bus Arbiter Core" I*n Proceedings of the 37th Design Automation Conference (DAC)*, pages 196-200, 2000.

[12] A. Roychoudhury, T. Mitra, and S. R. Karri, "Using Formal Techniques to Debug the AMBA System-on-Chip Bus Protocol", *In Design, Automation and Test in Europe (DATE)*, pages 828-833, 2003.

[13] K. W. Susanto and T. F. Melham, "An AMBA-ARM7 Formal Verification Platform", In *International Conference of Formal Engineering Methods (ICFEM)*, pages 48-67, 2003.

[14] H. Amjad, "Verification of AMBA Using a Combination of Model Checking and Theorem Proving", Electronic Notes in Theoretical Computer Science, *Proceed- ings of the 5th International Workshop on Automated Verification of Critical Systems* (AVoCS 2005), 145:45-61, 2006.