Design Space Exploration for Hardware/Software Codesign of Multiprocessor Systems

A. Baghdadi, N-E. Zergainoh, W. Cesario, T. Roudier†, A.A. Jerraya

TIMA Laboratory
46 avenue Felix Viallet 38031 Grenoble France
Amer.Baghdadi@imag.fr

† Arexsys, R&D center 1 Chemin du Pré Carré
38240 Meylan France

Abstract

In this paper, we present a new methodology to rapidly explore the large design space encountered in hardware/software systems. The proposed methodology is based on a fast and accurate estimation approach. It has been implemented as an extension to a hardware/software codesign flow to enable the exploration of a large number of multiprocessor architecture solutions from the very start of the design process. The effectiveness of this approach is illustrated by a significant application example.

1 Introduction

The ever growing demand of application performance makes multiprocessor architectures become more and more important in many industries (e.g. telecommunications). So in order to deal with these complex architectures and to meet the more severe time-to-market constraints we need new system design methods. Hardware/software codesign has emerged as a promising approach to cope with this challenge. One of the most important issues of this approach is design space exploration. In other words, it is important to find the best system architecture including the right partition between hardware and software components and the right hardware components and communication protocols. Starting from the same system specification, several architectures may be produced. The exploring of all these architectures requires the ability to rapidly determine the performance resulting from a particular partitioning.

The number of solutions for mapping a system specification made of n tasks (i.e. processes) on an architecture made of q nonempty modules (i.e. processors) may be computed using the stirling numbers of the second kind [1]:

$$S(n,q) = \sum_{i=0}^{q} \frac{(-1)^i \binom{q}{i} (q-i)^n}{q!} \qquad (1)$$

Now, we assume that we have p different kinds of technologies to implement each module. Each module may be implemented as specific hardware or targeted as a software executed on a specific processor. The following equation gives us the number of architectural solutions:

$$Nb_{Architecture}(n,p) = \sum_{q=1}^{n} p^q S(n,q) \qquad (2)$$

We notice that the number of solutions increases exponentially with n and p. For example, assume that we have a system composed of 4 tasks, now if we use 3 kinds of technologies, e.g. hardware implementation and two kinds of processors to execute software, we find a design space with 309 different architectures. This space will be even bigger, if we consider different communication protocols.

For each architecture, synthesis and low-level cosimulation may take days. Thus, we cannot afford to synthesize and to simulate at the cycle level every single architecture to measure its performance. These facts constitute the basis of our motivation for the work presented in this paper. They explain the need for a performance estimation approach which can accomplish the complex task of architecture exploration within a reasonable lapse of time. The combination of such an approach with a codesign flow constitutes a complete environment for the efficient implementation of complex heterogeneous multiprocessor systems.

1.1 Related works

In the literature, several works address architectural exploration and performance analysis within the hardware/software codesign context. Existing works on performance analysis for hardware/software codesign space exploration can be classified in two classes according to the complexity of the target architecture.

In the first class, the target architecture is monoprocessor. PMOSS [2], COSYMA [6], and LYCOS [10] follow this scheme. In PMOSS [2], the authors only calcu-

late the speed-up due to the coprocessor (i.e. hardware) on the overall system performance. In COSYMA [6], the authors calculate separate metrics for the software, the hardware and the communication parts. Then, these metrics are combined into equations to work out a partition based on the simulated annealing method. The communication time is assessed for their particular model: shared memory. In LYCOS [10], the authors estimate performance using profiling techniques and evaluations of low-level execution time for hardware, software and communication.

In the second category, the target architecture is multiprocessor. SpecSyn [3], POLIS [11], and the method proposed by Yen et al [12] follow this scheme. In SpecSyn [3], the authors deal with multiprocessor architectures. The performance estimation approach is mixed: static/dynamic. Yet, it is difficult to capture dynamic changes of the execution time during the design space exploration, as during this phase only simple static methods are used.

The approach for timing analysis used in POLIS [11] may capture much of the dynamic timing behavior because this system uses a combination of high-level simulation and low-level estimations (i.e. static/dynamic approach). This approach is similar to ours, but the target architecture of POLIS is composed of only one microprocessor and several custom coprocessors. The authors in [12] analyze at the system level the interaction between the different processes giving the best and the worst-case execution time for each of them.

None of the above works solve the problem of accurate estimation or hardware/software architecture exploration in the case of multiprocessor architectures. The main contribution of this paper is to provide an accurate performance estimation method enabling design space exploration in the case of hardware/software codesign. Our estimation/exploration methodology makes use of an existing system-level simulation tool and a codesign tool.

1.2 Methodology overview

When defining the specifications, one of the major issues of our methodology was the optimal trade-off between speed and accuracy. For accuracy, it is necessary to use simulation at the cycle level for every data-dependent behavior. However, this approach is not feasible in our context as it does not allow for fast exploration of the design space. This is why we rather use simulation at the system level. This high-level simulation allows us to evaluate the dynamic behaviors of the interaction between the different processes, whatever the complexity of the architecture. However, it lacks accurate timing information. To make up for this shortcoming, we use, in addition, a back-annotation approach. As a matter of fact the analysis of "some" implementations (at RT level) allows us to extract "all" timing elements needed for performance estimation of "all" feasible implementations. These timing elements are then combined and introduced into the system specification once and for all. Thanks to this new time-annotated specification it is possible to predict the performance of all feasible architectures.

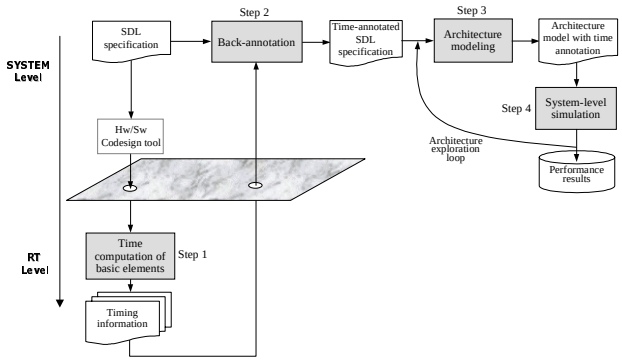

Figure 1. Estimation/exploration methodology

The estimation/exploration methodology we present in this paper makes use of the system-level simulator GEODESIM from the ObjectGEODE environment of VERILOG [9] and the codesign tool MUSIC [8]. The system design flow starts with a specification given in SDL [7].

We can identify four main steps in the overall flow of our methodology (see Figure 1): Time computation of basic elements, back-annotation, architecture modeling and system-level simulation. These steps will be explained in the fourth section in detail.

The rest of this paper is organized as follows. Section 2 presents some aspects of the tools we used. Section 3 describes the proposed estimation/exploration methodology. Section 4 contains the experimental results of our method applied to an application example. Finally, section 5 concludes the paper.

2 System design environment

The codesign tool For this work, we used a codesign tool called MUSIC [8]. It starts from the system-level specification language SDL to produce heterogeneous multiprocessor architectures composed of hardware and software components. The codesign flow may be summarized to three main stages as shown in Figure 2:

— *System modeling*: In this stage, the required system functionality is specified. The system is described in SDL [7]. This specification is validated by the use of the GEODESIM simulator [9].

— *Partitioning and communication synthesis*: This stage examines design alternatives to identify those alternatives that meet the system constraints. It consists in mapping the functions of the system onto interconnected hard-

ware/software processors. At this stage, the main architectural choices are made and tools are required to estimate the performances of the architectures.

— *Prototyping*: This stage carries out the generation of the prototype. It is divided into two intermediate steps. The first step consists in generating a functional prototype (VHDL/C) which may be validated through cosimulation [5]. The second step consists in targeting the components of the architecture and generating a cycle accurate model, which may also be validated through cosimulation [5].

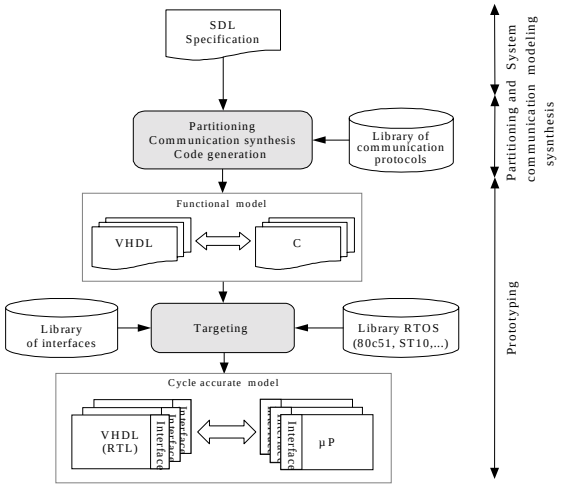

Figure 2. The codesign flow

GEODESIM simulator The GEODESIM simulator [9] employs the semantics of the SDL language (Z.100 and Z105 recommendations) [7]. Additionally, it uses an extra annotation format for the assessment of performances and architecture modeling [9]. These annotations have no influence on the initial specification of the system since they are considered as comments and only interpreted by the simulator. This is possible thanks to the so-called directives i.e. COMMENT strings, that contain a specific pattern that is recognized by the GEODESIM simulator. These directives can be attached to individual actions inside the SDL transition in order to get accurate evaluations. Figure 3 illustrates the architecture-modeling directives:

— The NODE directive allows to identify the execution resources of a given model, called nodes. This is generally used to specify the kind of processor that will be used to execute the module. For instance the first node in Figure 3 is attached to a 80C51 processor. A node can be associated to systems, blocks or processes. All processes (or blocks) inside a node share the same execution resources. Processes, which are not inside a node, are considered as default nodes.

— The PRIORITY directive allows to assign an order of priority to every SDL process, inside the same node, to model priority-based multitask execution (modeling of the scheduling strategy). In standard SDL, the choice of a transition among all fireable transitions of all process instances inside a node is done according to a random uniform distribution. The PRIORITY directive can be associated with a process in order to modify this random choice.

— The DELAY directive is associated with SDL actions in order to specify their execution time. The default execution time of an action is zero delay. During the simulation, when a delay action is reached, the corresponding node is blocked during the time specified by the directive parameters, then the action is executed and its effects can be observed.

```
1  SYSTEM   S1;
2        …
3        BLOCK  b1   COMMENT  '#NODE  80C51';
4               …
5                    PROCESS  p1  COMMENT   '#PRIORITY   (1)';
6                    …
7                    TASK    COMMENT   '#DELAY   ((5+mult1+3) * p_8051)';
8                    TASK    a := b*c;
9                    …
10                   ENDPROCESS;
11                   PROCESS  p2;
12                            ...
13       ENDBLOCK;
14
15       BLOCK  b2   COMMENT  '#NODE  ST10';
16              ...
17       ENDBLOCK;
18 ENDSYSTEM;
```

Figure 3. Architecture-modeling directives

3 Estimation/exploration methodology

The proposed methodology is based on an accurate estimation approach and has been implemented as an extension to MUSIC codesign flow. It takes advantage of both system and RT levels of abstraction, and combines both static and dynamic analysis techniques, in order to obtain the best trade-off between speed and accuracy. The facts that delays are computed starting from the implementation model (at RT level) and that dynamic behaviors are captured by simulation (i.e. dynamic analysis) allow to have a very high accuracy. The speed is maintained by the fact that the architecture exploration loop (see Figure 1) is performed at the system level (using system-level simulation). The analytic aspect of our approach is noticeable at the computation of the timing elements, the back-annotation equations, and the architecture modeling stages.

In this section, we will explain the four stages of the flow of our methodology in detail (see Figure 1).

3.1 Time computation of basic elements

This first stage consists in analyzing an implementation (at RT level), generated by MUSIC, to calculate the execution time of all SDL operations (i.e. computation and communication operations).

Each SDL process may have a software or a hardware realization. For the software realization, the SDL-specified process will be translated into C code. The execution time

computation, which depends on the couple "processor, compiler", will be achieved by the analysis of the corresponding assembler code. For the hardware realization, the SDL process is translated into VHDL RTL code. At this abstraction level we assume that parallelism is limited to the cycle level, and we consider a simple architecture model: each VHDL RTL transition takes one clock cycle.

The first step consists in identifying and labeling of the beginning and the end of each basic block in the SDL specification. A basic block – noted BB – is a sequence of operations without control instructions (branches and conditioning) or communication. The correspondence between SDL actions and the generated code is obtained thanks to this labeling and by a technique worked out for the code generator of MUSIC. After identifying the basic blocks in SDL and starting from the same system specification, two implementations are made: one purely software and the other one purely hardware. The software implementation may give several realizations according to the number of available couples "processor, compiler". The generated codes (assemblers, VHDL RTL) are analyzed to isolate the basic blocks and to calculate their execution time in terms of "number of clock cycles". Figure 4 gives the overall flow.

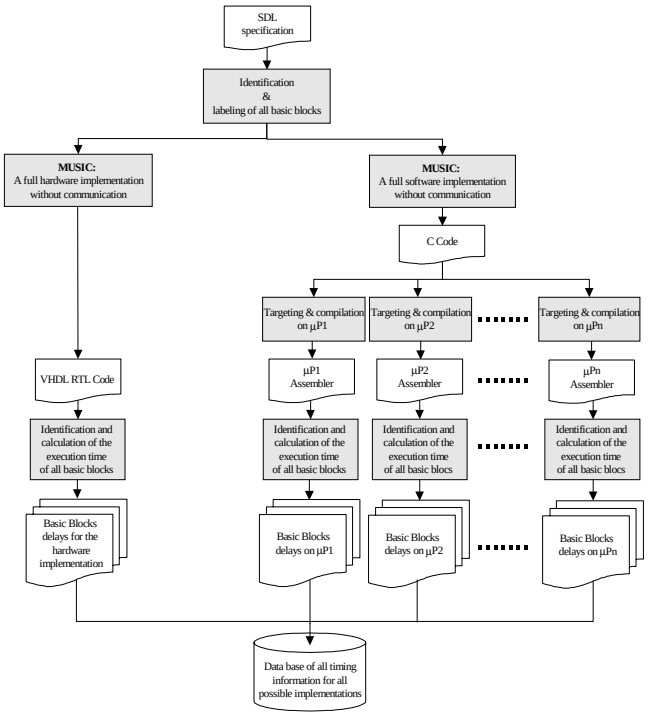

Figure 4. Time computation of the basic blocks

Modeling the communication time and synchronization between two processes actually means modeling three kinds of delays [4]: interface initialization time (particularly for external communication), $T_{Startup}$; data transmission time, T_{Trans}; and synchronization time, $T_{Synchro}$. Thus, the time needed for a communication operation can be modeled by the following equation:

$$T_{Comm}(n) = \lambda T_{startup} + T_{Trans}(n) + T_{Synchro} \qquad (3)$$

where *n* is the amount of data (e.g. number of bytes), and λ stands for "0" or "1" according to the type of internal or external communication respectively. Interface initialization time, $T_{Startup}$, is the fixed part of the protocol that remains independent from the amount of data. For a particular communication protocol, this time only depends on the target processor. Data transmission time, T_{Trans}, depends on the amount of data and the communication speed. Synchronization time, $T_{Synchro}$, is the time the process is delayed for the communication to be completed.

In fact, as $T_{Synchro}$ is execution-dependent it is taken into account by the GEODESIM simulator thanks to its supported communication schemes [9]. $T_{Startup}$ and T_{Trans} are calculated for each communication protocol following the same computation flow as the one shown in Figure 4. Yet, the starting point is the description of the communication protocol in SDL or in another intermediate language supported by MUSIC.

Unlike the execution time computation of the basic blocks, which has to be done again and again for every new application, the communication time can be reused in other applications. The results of this stage are used to build a database containing enough timing information for the architecture exploration loop (cf. Figure1).

3.2 Back-annotation

During this stage, the SDL specification is annotated with the timing elements calculated during the previous stage using therefore the directive DELAY.

For every basic block in SDL, we got several delays corresponding to the different possible realizations. The timing parameters corresponding to these delays are inserted in SDL by adding "TASK COMMENT DELAY ExecTime_BB(x)_Techno(p)" the beginning of every basic block. The parameter "ExecTime_BB(x)_Techno(p)" is replaced by a numerical value at the architecture modeling stage according to the selected realization. When the simulator reaches the operation "TASK COMMENT DELAY ExecTime_BB(x)_Techno(p)", the process is blocked during the time specified by ExecTime_BB(x)_Techno(p), while the global time is progressing.

Concerning communication, an action that models the communication time is inserted in SDL by adding "TASK COMMENT DELAY $T_{Comm}(n)$" after all the *output* and *input* actions in the communicating processes. The parame-

ter $T_{Comm}(n)$ is given by the equation 3 (cf. §3.1), and it is replaced by numerical values at the architecture modeling stage according to the selected protocol and using the library of communication delays.

3.3 Architecture modeling

The SDL specification obtained from the previous stage is instrumented with the basic timing elements of all available technologies. Then, in this stage, the designer has to instrument this specification with additional information to target one specific system architecture. The result of this stage is an instrumented SDL specification that fully models the timing behavior of the execution of the selected architecture.

The architecture information that the designer has to introduce in the SDL specification is related to the system partitioning and technologies used for the design of each module. This may be achieved by three steps:

1. *Selection of SDL processes that will share the same execution resource*: It consists in modifying the hierarchical structure of the SDL model in order to gather all processes that have to be executed on the same processor in the same SDL block. Then the directive NODE is added to the declaration of this new block

2. *Assignment of the software and hardware processors*: In this stage of processor assignment, the timing parameters of all the basic blocks are replaced by the numerical values corresponding to the target processor.

3. *Selection of the communication protocols*: During this stage communication protocols are selected and associated with communication channels and actions in SDL using the library of communication delays.

3.4 System-level simulation

This last stage consists in executing the architecture-annotated SDL specification using the GEODESIM simulator. The simulation result gives us the performances of the architecture. We define the same test-bench for all the architectures that we have to compare. At the end of every simulation, we collect the overall execution time.

With the GEODESIM simulator, the scheduling of all the processes of a node is done according to a uniform random distribution. The directive PRIORITY associated with a process allows us to change this order of execution.

4 Results

In this section, we present the results that we obtained from the design of a robot arm control system. We started by describing the system functionality in SDL. Then, several architectural solutions were explored using our estimation/exploration methodology. These architectures were also synthesized using an existing codesign tool called MUSIC [8]. The resulting architectures were validated by cycle accurate cosimulation. The comparison of these two results shows the effectiveness of our approach.

4.1 Robot arm control system

This system adjusts the speed variation of each motor for the arm to be smoothly moving and for the physical constraints of acceleration and braking to be met. The complete version of this application contains 18 motors. In our case, we only consider two motors for a clear explanation of the example. The SDL specification of the system is composed of four processes as shown in Figure 5. The process HOST sends speed and control parameters to the PID in order for the latter to take over the control of the two motors.

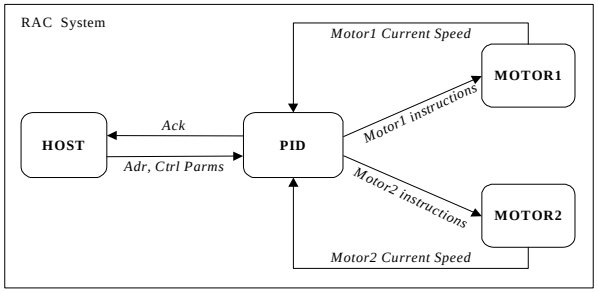

Figure 5. Robot arm control system.

4.2 Method assessment

Three kinds of technologies are available: two microcontrollers — ST10 and 80C51 — and the hardware option. Concerning communications, we limited our choice to a point-to-point communication with a “Rendez-vous” protocol. For this sample example, and using the equation 2 (cf. §1), we find a design space with 309 different architectures. Of course, this number will drop significantly if we consider some non-functional constraints (e.g. the two motors are always implemented with the same technology).

We applied the different stages of our approach. Among all the architectures of the design space, we selected five alternatives to explore. We used the same test-bench for all the architectures to compare their performances. In addition, in order to validate the results obtained with the estimation/exploration method, we measured the performances of the five architectures generated through synthesis and RT-level cosimulation using the MUSIC codesign tool. The partitioning and the assignment of processors for these architectures, as well as their performances are illustrated by Figure 6. The given time corresponds to one control iteration.

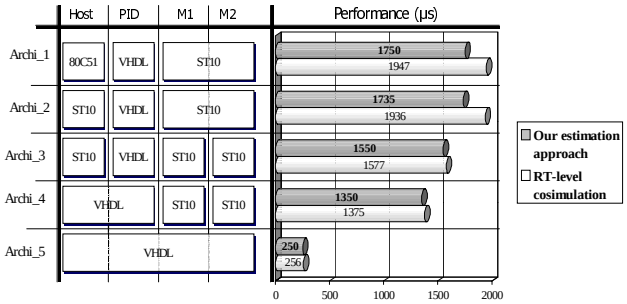

Figure 6. Performances vs. architectures

4.3 Analysis of the results

The comparison between the performance results obtained with our estimation approach and those obtained through synthesis and RT-level cosimulation is given in Table 1.

Table 1. Error rate of the estimation approach

Architectures	Archi_1	Archi_2	Archi_3	Archi_4	Archi_5
Error Rate	10,1%	10,4%	1,7%	1,8%	2,3%

We notice that the results are almost perfect for the last three architectures and lack precision when the two processes of the two motors share the same processor (e.g. ST10). The execution of these two processes on the ST10 is sequential. However, the scheduling strategy of the execution model of the GEODESIM simulator is set to random. This lack of precision was quickly fixed through the implementation of a new predefined scenario modeling the right sequential execution of both processes in the simulator. With this transformation, the error rate, in the case of the first two architectures, drops from 10% to about 3%. These very small error rates for all the explored architectures show the high accuracy of our approach.

Considering the simulation speed, the result is even more promising since the simulation time at the system level of an architecture model is 10^4 times faster than the cosimulation of the same architecture at the RT level. The system-level simulation of each of the five architectures takes approximately 10 seconds, whereas the cycle accurate cosimulation takes more than 30 hours.

From a practical point of view, one of the most important advantages of our methodology is the facility to automate all its stages. Actually, this methodology is based on the codesign flow of MUSIC and GEODESIM simulator. Yet, the principal ideas can be reused to adapt this methodology to other system design environments.

5 Conclusion

This paper addressed performance estimation and architecture exploration issues within the context of hardware/software codesign. We developed and illustrated the possibilities of a new estimation/exploration methodology. It enables the exploration of a large number of multiprocessor architecture solutions from the very start of the design process. The effectiveness of this approach was illustrated by a significant application example. Experimental results indicate strong advantages of the proposed methodology.

Acknowledgments

This work was supported by France-Telecom/CNET, STMicroelectronics, ESPRIT program under project COMITY 23015 and MEDEA program under project SMT AT-403. We also acknowledge M.Y. Teruya for his help during this work.

References

[1] Abramowitz, M.; Stegun, C. A. (Eds.): *Stirling Numbers of the Second Kind*. in «Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables» 9th printing, New York: Dover, pp. 824-825, 1972.

[2] Eikerling, H.J.; Hardt, W.; Gerlach, J.; Rosenstiel, W.: *A Methodology for Rapid Analysis and Optimization of Embedded Systems.* International IEEE Symposium and Workshop on ECBS, D-Friedrichshafen, pp. 252-259, March 1996.

[3] Gajski, D.; Vahid, F.; Narayan, S.; Gong, J.: *System-Level Exploration with SpecSyn*. Design Automation Conference, pp. 812-817, June 1998.

[4] Gajski, D.: *Principles of Digital Design*. Prentice Hall, Upper Saddle River, NJ, 1997.

[5] Hessel, F.; Le Marrec, P.; Valderrama, C.; Romdhani, M.; Jerraya, A.A.: *MCI: Multilanguage Distributed Cosimulation tool.* DIPES 98, Paderborn, Germany, October 1998.

[6] Henkel, J.; Ernst, R.: *High-Level Estimation Techniques for Usage in Hardware/Software Co-Design*. Asia and South Pacific Automation Conference, pp. 353-360, Yokohama, Japan, February 1998.

[7] ITU-T *Functional Specification and Description Language*. Recommendation Z.100 - Z.104, March 1993.

[8] Jerraya, A. A. et al.: *Multilanguage Specification for System Design and Codesign.* Chapter in «System-level Synthesis», NATO ASI 1998 edited by A. Jerraya and J. Mermet, Kluwer Academic Publishers, 1999.

[9] Le Blanc, P.: *SDL Performance Analysis with ObjectGEODE Verilog white paper*. VERILOG, Toulouse, France 1998.

[10] Madsen, J.; Grode, J.; Knudsen, P.V.; Petersen, M.E.; Haxthausen, A.: *LYCOS: the Lyngby Co-Synthesis System.* Journal for Design Automation of Embedded Systems, Kluwer. vol.2, no.2, pp. 195-235, March 1997.

[11] Suzuki, K.; Sangiovanni-Vincentelli, A.: *Efficient Software Performance Estimation Methods for Hardware/Software Codesign*. Design Automation Conference, June 1996.

[12] Yen, T-Y.; Wolf, W.: *Performance Estimation for Real-Time Distributed Embedded Systems.* International Conference on Computer Design, 1995.