Software Performance Estimation in MPSoC Design

Marcio Oyamada[1, 2] Flávio R. Wagner[1] Marius Bonaciu[2] Wander Cesario[3] Ahmed Jerraya[2]

[1] UFRGS
Instituto de Informática
Porto Alegre, RS, Brazil
{marcio, flavio}@inf.ufrgs.br

[2] TIMA Laboratory
SLS Group
Grenoble, France
{marius.bonaciu, ahmed.jerraya}@imag.fr

[3] MnD (Methodologies & Designs),
Montigny-Le-Bretonneux, France
wcesario@mnd.fr

Abstract - Estimation tools are a key component of system-level methodologies, enabling a fast design space exploration. Estimation of software performance is essential in current software-dominated embedded systems. This work proposes an integrated methodology for system design and performance analysis. An analytic approach based on neural networks is used for high-level software performance estimation. At the functional level, this analytic tool enables a fast evaluation of the performance to be obtained with selected processors, which is an essential task for the definition of a "golden" architecture. From this architectural definition, a tool that refines hardware and software interfaces produces a bus-functional model. A virtual prototype is then generated from the bus-functional model, providing a global, cycle-accurate simulation model and offering several features for design validation and detailed performance analysis. Our work thus combines an analytic approach at functional level and a simulation-based approach at bus functional level. This provides an adequate trade-off between estimation time and precision. A multiprocessor platform implementing an MPEG4 encoder is used as case study, and the analytic estimation results in errors only up to 17% when compared to the virtual platform simulation. On the other hand, the analytic estimation takes only 17 seconds, against 10 minutes using the cycle-accurate simulation model.

I. INTRODUCTION

Advances in technology provide the development of complete multiprocessor systems integrated on a single chip (MPSoC), including heterogeneous processors, application-specific HW components, memories, digital interfaces, and occasionally analog interfaces.

MPSoC complexity demands new system-level tools that support the design above the RT-level. Performance is usually one of the main criteria adopted to guide the architectural design. However, other aspects also need to be evaluated as soon as possible in the design flow, such as power, energy, and area. Considering the huge design space, new system-level methodologies have to support a fast and flexible design space exploration. This requires high-level performance estimation tools integrated with exploration strategies in order to help the ranking of design alternatives.

Flexibility, time-to-market, and cost requirements have made software a dominant part of current embedded systems. Therefore, high-level software performance estimation tools are needed in early steps of an MPSoC design flow. Although software performance estimation gives basic information about the system performance, it does not consider some system-wide effects, such as those imposed by communication mechanisms in a multiprocessor environment. This requires an integrated HW/SW estimation approach that considers these inter-component relationships.

Many academic and commercial tools are provided for software performance estimation. Analytic software performance estimation, however, is an open research topic. Proposed approaches aim at the development of fast and accurate methods, but they are not usually integrated in a complete design flow. After the design refinement, some approaches propose global simulation models that allow the complete MPSoC performance analysis. However, these simulation platforms are not linked in a global design flow, often requiring manual modeling of the virtual prototype for each design.

In this paper, an appropriate methodology for performance analysis at different abstraction levels, offering different trade-offs between estimation speed and accuracy, is tightly coupled with a synthesis environment that provides a path to implementation. A software performance estimation technique based on neural networks is used, targeting at the processor selection [1]. At a high abstraction level, this estimator provides fast results and helps the designer select the suitable processor to run a given application. The processor selection is then used in an MPSoC design environment for a further HW/SW refinement [2], offering a path to the final implementation. This refined architecture is evaluated by means of a virtual prototype based on cycle-accurate simulation, providing an integrated design and performance evaluation environment. In the case study presented in this paper, estimation errors of only up to 17% when compared to virtual platform simulation results have been found, while the estimation process takes only 17 seconds, against 10 minutes executing the cycle-accurate simulation model.

The remaining of this paper is organized as follows. Section 2 discusses related work. Section 3 presents an integrated methodology for performance estimation and MPSoC design. Section 4 presents the analytic neural network estimator, and Section 5 describes the performance analysis based on a virtual prototype. A case study of an MPEG4 encoder is presented in Section 6, and Section 7 draws the conclusions and perspectives.

II. RELATED WORK

A. Software performance estimation

Software estimation tools can be divided in two groups: simulation and analytic-based. Simulation-based methods use cycle-accurate simulators to estimate the software execution time. Analytic methods use abstract models and cost functions to calculate the software execution time. An intermediate solution uses code annotations (at instruction or basic block level) with the execution cost to estimate the software performance [3]. In this case, the application runs natively, thus overcoming the large execution time of cycle-accurate simulators.

Analytic software performance estimation methods are proposed to provide a fast estimation with a low modeling and execution effort. This is useful for high-level design space exploration. Usually, an application profiling is performed to extract the number of executed instructions of various types [3,4,5,6]. After this, a method maps these instructions to a performance model that computes the execution time.

Giusto et al. [4] compile the application code into a virtual instruction set (a simplified RISC set with 25 instructions). The estimation is performed by evaluating the execution cost of the virtual instructions in the target architecture. The cost is estimated using a training set and applying a linear regression method. The authors show that, due to the linear approximation, the proposed method is accurate only when the training set and the application are similar.

Bontempi and Kruijtzer [5] use a nonlinear method to estimate the execution time. For a given benchmark set, a profiler extracts a functional signature vector for a virtual processor (with a set of 42 instructions), containing the instruction types that appear in the code and the number of times each instruction type is executed. They also use a training approach to calibrate the estimator. In the utilization phase, they apply a modeling technique called lazy learning to choose an estimation function that is based on a criterion of neighbourhood between the application and the training set. This function, which may be locally linear, uses only points of the training set that are closer to the application. They report a mean error of 8.8% in the estimations, for a set of 6 benchmarks, each one executed with 15 different input data sets.

In this work we propose a non-linear estimator based on neural networks that is more precise than linear methods such as proposed by Giusto [4], mainly for advanced architectures. Our method is similar to that proposed by Bontempi [5], but in our case we use the instruction set of the target processor, resulting in a better instruction classification and consequently increasing the precision.

B. *Performance estimation with virtual prototypes*

In a refined architecture, where the processor is already selected and hardware and software components have been defined, a global simulation model may be used for performance analysis. In MPSoC architectures composed of multiple processors, hardware IPs, memories, and peripherals, the evaluation of individual components is not sufficient to analyze the system performance. A virtual prototype provides a global simulation model, integrating a cycle-accurate model of the target processor with simulation models of the further hardware modules, described either at TLM or RT-level.

Currently, virtual prototype environments for modeling and simulation based on SystemC, such as MaxSim [7], Coware ConvergenSC [8], and Synopsys System Studio [9], provide a rich set of components such as processors, memories, and peripherals that can be extended by user-defined modules. Using these components, an MPSoC platform is modeled and simulated. Additional tools support the RTL synthesis for given components of the library, thus providing an automatic path to the silicon.

MPARM [10] is an environment for MPSoC design space exploration using SystemC. It is a complete platform solution for MPSoC simulation composed of processor models (ARM), bus models (AMBA), memory models, hardware support for SMP (hardware semaphores), and a software development toolset including a C compiler and an operating system (uCLinux). A cycle-accurate instruction set ARM simulator developed in C++ is encapsulated in a SystemC wrapper and integrated into the platform. The wrapper implements the interface and synchronization between the ISS and the SystemC simulation framework. This integration allows one to plug the ISS into a system simulation, activated by a common system clock, thus providing a consistent and synchronized hardware and software multiprocessor simulation. MPARM provides several performance statistics, such as cache miss/hit rate and bus contention and average transfer waiting time.

Meyr et al. [11] propose a link between processor models generated from the LISA architecture description language and SystemC-based simulation. Processor models are described in LISA either at instruction or cycle-accurate level. The rest of the system, including buses, IP components, and memories, is described in SystemC. The goal is to jointly explore processor and communication using a system-level approach. An integrated co-verification environment provides a way to analyze the software-related performance, such as CPU load and RTOS overhead. Furthermore, shared resources (for example memory and buses) directly affect the SW performance, and an isolated analysis of a single processor would hide potential problems and bottlenecks.

In virtual prototype environments, the architecture development starts with the virtual prototype modeling. Usually, a more abstract level is not supported in these environments. Virtual prototypes build a global simulation model and support performance analysis. However, the integration with the design flow is poor, and many manual configurations are needed. In this work, the virtual prototype is integrated with a hardware and software refinement design tool. This integration allows one to start the design at a high abstraction level and enables the automatic generation of the virtual prototype. This work uses the MaxSim [7] environment as virtual prototype environment, due to the support for SystemC models and predefined support for performance analysis.

III. Integrated Methodology for MPSoC Design and Performance Analysis

In this section, an integrated methodology for design and performance analysis of an MPSoC is presented. This methodology is proposed to support software performance estimation, mainly for the processor selection or evaluation at functional level, and global performance analysis using a virtual prototype. Other estimation tools that may be necessary to guide additional aspects of the MPSoC architecture exploration, such as HW/SW partitioning, task partitioning, or communication interconnection design, can be easily integrated to the proposed methodology.

Figure 1(a) presents the first step in our software performance estimation methodology. After the partitioning between hardware and software components, each software component needs to be mapped into a given processor. Our neural network (NN) estimator aims at the evaluation of this processor selection process. This methodology is adequate to drive the mapping of a task to a predefined portfolio of processors, where a trained NN estimator is available for each processor.

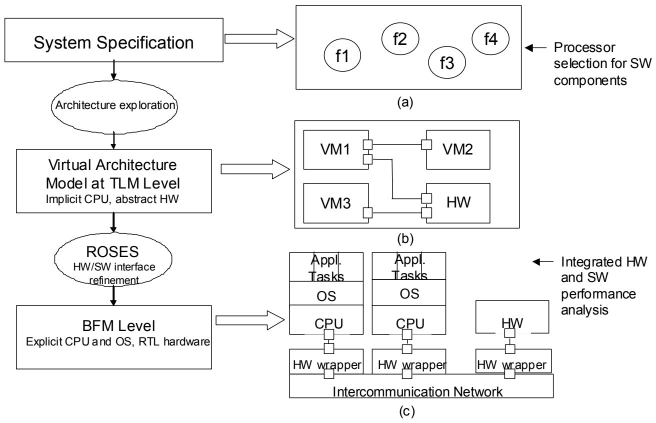
System Specification
Architecture exploration
Virtual Architecture Model at TLM Level
Implicit CPU, abstract HW
ROSES
HW/SW interface refinement
BFM Level
Explicit CPU and OS, RTL hardware
f1
f2
f3
f4
Processor selection for SW components
(a)
VM1
VM2
VM3
HW
(b)
Appl. Tasks
OS
CPU
HW
HW wrapper
Intercommunication Network
Integrated HW and SW performance analysis
(c)

Fig. 1. Integrated methodology for MPSoC design and performance analysis

After the high-level architecture exploration, a virtual architecture composed of hardware and software components and TLM channels for communication is derived, as shown in Figure 1(b). Software components are organized in tasks and use a communication API. Hardware components are considered as IP blocks, and only their interfaces are known. The virtual architecture is used to validate the software functionality and the communication between the components.

The virtual architecture is used as input for the refinement of HW and SW interfaces [2]. Software interfaces include all device drivers required to implement the communication API and a dedicated operating system for each processor. The HW/SW interface refinement follows a component-based approach. The interfaces are assembled using basic elements that implement the services required by the application. These elements are configured with adequate parameter values. For software components, parameters include for instance the processor type, and for the channels they include the communication protocol (e.g. FIFO or handshaking).

After the refinement of HW and SW interfaces, a bus-functional model (BFM) architecture is derived, as shown in Figure 1(c). In this architecture, the software part is modeled as tasks running on an operating system. Hardware components are available at RT-level as IP components. A virtual prototype, for estimation purposes, is generated from the BFM architecture as input to the MaxSim [7] environment. An instruction set simulator is used as processor model, and SystemC modules are used for hardware components.

The performance analysis at this level enables the designer to jointly verify the SW and HW. The designer may validate design decisions such as scheduling policies, drivers, and buffer sizes. Using a virtual prototype, the designer can also verify the impact of different cache sizes and memory hierarchies in the final performance. Profiling results, such as the execution time of each function, make possible the optimization of the software code.

IV. SOFTWARE PERFORMANCE ESTIMATION BASED ON NEURAL NETWORKS

High-level performance estimation is an interesting alternative, since it may combine a low cost for obtaining the performance data with an acceptable precision. This allows a fast evaluation of different processor alternatives in the early phases of the design cycle. The main problem to develop a software estimation tool is an accurate performance model that considers advanced architectural features such as pipelines, caches, and branch predictors.

The exact number of cycles required by an application may be obtained using the real processor or a cycle-accurate simulation. These techniques, however, have an inherent high cost for the development and setting of the simulation model.

Neural networks have been chosen for performance estimation since they can generalize their behavior even when the process to be modeled is highly non-linear, which is the case of software running on processors having pipeline and cache effects. In this work, a feedback-forward network [12] has been used, due to its simplicity and adaptation to the non-linear behavior of software performance estimation.

Figure 2 presents the two main steps of our estimation method: training and utilization. In the training phase, a set of samples is presented to the network. Its inputs are the number of executed instructions of different instruction types (branches, integer and floating point arithmetic, memory accesses, etc), while the expected result is the number of cycles consumed by the application.

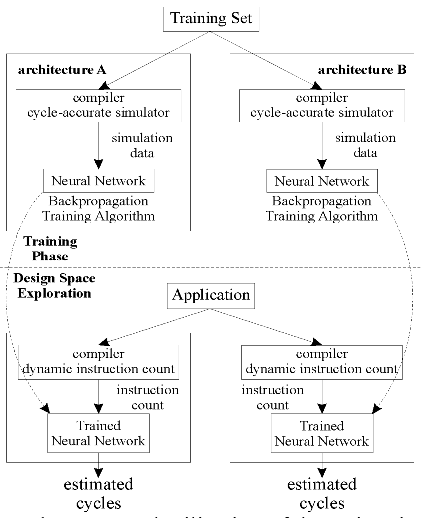

Fig. 2. Development and utilization of the estimation tool

Figure 3 presents the neural network used to estimate the application cycle count for the ARM processor, where the inputs are the number of instructions of the different types. It is composed by an input layer, a hidden layer with 5 neurons containing a tansig transfer function, and an output layer with one neuron containing a linear transfer function. These transfer functions are available in the Matlab Neural Network Toolbox [13].

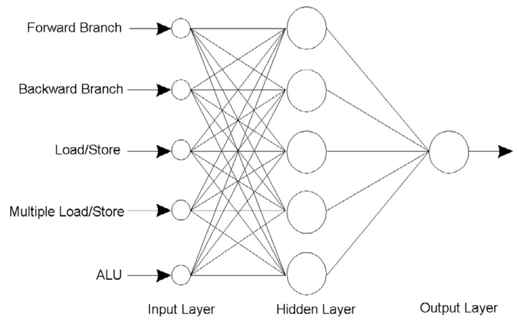

Fig. 3. NN performance estimation

For training, a cycle-accurate simulator is required to extract the number of executed instructions and the total number of cycles consumed by a set of training benchmarks. We have selected a small number of instruction classes that are sufficiently representative of the timing behavior of all instruction types (forward branch, backward branch, load/store, multiple load/store, and ALU). An iterative learning process, based on the back-propagation algorithm, modifies the weights of the input and output arcs of neurons in each layer, so the network presents an output that is as close as possible to the expected result. The training phase is realized using the Matlab software. After the training phase, the estimation tool is ready to be used in many designs.

In the utilization phase, an application is compiled for the given target processor, and the number of executed instructions of each type is obtained by a dynamic instruction count and presented to the neural network, so that it can estimate the number of cycles consumed by the application. An alternative approach to obtain the number of executed instructions is a static method, as proposed in [14,15].

The training time may be long, depending on the inputs and complexity of the generalization. However, once the network is trained, its utilization has a low cost, consisting in the dynamic instruction count of the application and in the neural network cost, which requires only the multiplication of the inputs by the weights of the neurons. The dynamic instruction count dominates the time consumed in the utilization phase, but it is faster compared to a cycle-accurate simulation, as presented in Table III (Section VI).

For each target processor, a different estimator is generated. This performance estimation method is especially adapted for evaluating if a candidate processor can execute a certain application or task under given performance requirements. It is also adequate for design space exploration in the software domain, for instance considering various algorithmic alternatives for design tasks and various partitionings of tasks among processors. Architectural modifications in the processor, however, would require a new training process and thus a long turnaround time.

V. Performance analysis using a virtual prototype

After the refinement of hardware and software interfaces, the bus-functional model (BFM) is used to generate a virtual prototype. The software part is composed of tasks that execute upon an operating system in each target processor. The operating system is responsible for implementing the API (application programming interface) used for the communication between components. Hardware components are described in SystemC. The evaluation of this virtual prototype is required before the physical design, giving to the designer detailed information about the overall system performance.

In this work, the MaxSim [7] environment is used to generate a virtual prototype model enabling performance evaluation. MaxSim is based on SystemC. This simulation model is automatically generated from the architecture description that includes the components, their interfaces, and the connections between them.

Hardware components are considered as IP blocks, for which it is supposed that cycle-accurate models are supplied. The hardware interface adapters generated in the HW/SW refinement step are also available as SystemC cycle-accurate models. The SystemC components are encapsulated in MaxSim components and added to its library.

Software is simulated using cycle-accurate processor models available in the MaxSim component library. They are integrated to the hardware simulation models, resulting in a single global simulation model. Hardware and software simulators run in a synchronized way, making possible the detection of problems arising from the communication between components, interrupt handling, and others.

At this level, performance evaluation allows the analysis of the influence of each component in the global performance. For the software analysis, the execution timeline gives the cycles consumed by each application function, easing the detection of bottlenecks and optimization points. Different cache sizes can be tested and their performance analyzed. MaxSim supports the custom profile of user-defined components and provides a profiling interface that allows the module instrumentation. The performance events generated during the simulation are visualized in XY charts.

VI. Case Study: MPEG4 encoder

In this section, an MPEG4 encoder platform is used as case study to show the application of the proposed methodology.

A. Application overview

We evaluate the proposed approach using a parallelized MPEG4 encoder platform [16], shown in Figure 4. The execution of the MPEG4 encoder application is divided among two processors. The first processor is responsible for the core encoder algorithm, while the second one implements the Huffman compression code (VLC). The architecture is flexible and allows the parallelization of the Encoder and VLC tasks. Input, Combiner, and Direct Memory Access (DMA) are hardware IP components. The Input component divides one frame among the parallelized Encoder processors, and the Combiner is responsible for merging the results from VLC processes. The DMA hardware component is responsible for managing the transfers among the components.

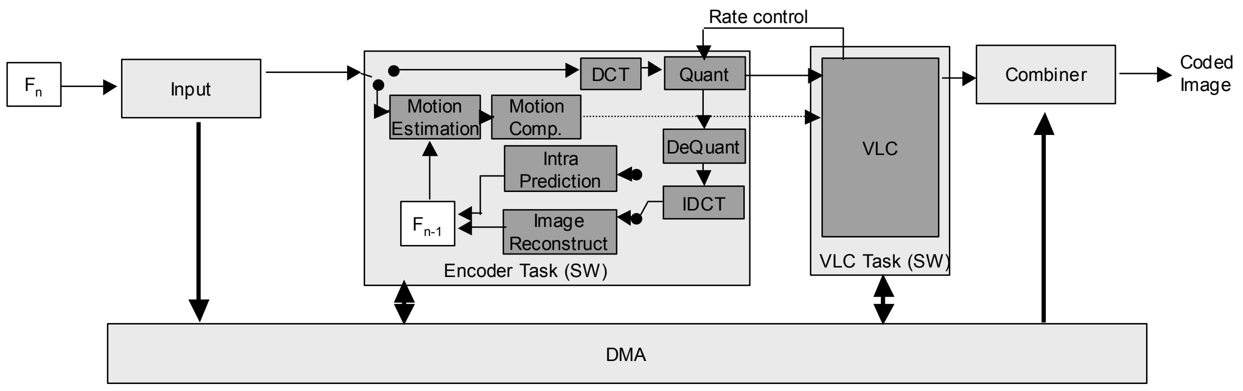

Fig. 4. Virtual architecture organization of the MPEG4 encoder

The parallelization allows the platform configuration for different encoder profiles, from low definition to high-definition video. Moreover, the platform allows the utilization of parallel processors running at low frequencies instead of a high-performance and high-power consumption processor.

We consider that the system specification has been already partitioned in terms of hardware and software components. However, the processors for the software execution need to be defined.

After the architecture exploration and definition of the virtual architecture, the HW/SW interface refinement is made based on the processor chosen in the previous step. Then, a virtual prototype is generated to evaluate the performance of the refined architecture.

B. Neural network estimation

The target processor to be evaluated was the ARM946 [7], with 4 Kbytes of data cache and 4 Kbytes of instruction cache. A set of benchmarks composed of control-dominated and data-dominated applications was used to train and test the estimator. The benchmark set is composed of 30 different benchmarks, with a total of 40 samples, because some benchmarks are used with different inputs. Table I describes the benchmarks used in the experiment.

TABLE I. BENCHMARK SET

Sort and Search	Quicksort, bubble sort, selection sort, sequential search, binary search
Numerical	Matrix multiplication, matrix inversion, matrix sum, matrix count, root computation, square root computation, LU decomposition, statistics (mean, variance, standard deviation), Fibonacci, complex number arithmetic operations
Data Processing	FFT, FIR, data compress, DES cryptography, ADPCM (Adaptive Differential Pulse Code Modulation), DCT (Discrete Cosine Transform), CRC (Cyclic Redundancy Check), LMS (least-mean square) algorithm
Synthetic	6 synthetic algorithms
Statecharts	Code generated automatically from Statecharts descriptions

To verify the precision of this estimator, a set of 18 samples was used in the training phase and the remaining ones as test set. The results are presented in Table II. The mean estimation error considering all benchmarks is 9.05%. In the test set, the mean error is 10.08% with maximum and minimum errors of 11.58% and –35.59%, respectively. Such errors, even high, are acceptable in a high-level performance estimation.

To obtain the dynamic instruction count used in the estimation step, an instruction-accurate simulator was used. Table III presents the speedup obtained with the NN estimation when compared against the cycle-accurate simulation. Even requiring a dynamic instruction count, the NN estimation is much faster due to the complexity and time required in the full processor simulation needed in the cycle-accurate simulation.

TABLE II. ARM946 ESTIMATOR RESULTS

	Min error	Max error	Mean error	Std deviation
All benchmarks	-35.59%	29.75%	9.05%	8.90%
Training set	-18.30%	29.75%	7.62%	7.97%
Test set	-35.59%	11.58%	10.08%	9.54%

Using the same neural network to evaluate the Encoder and VLC tasks, the estimated numbers of executed cycles are 122,910 and 21,613, respectively. This estimation is performed only for one macroblock of 16x16 pixels, since the same execution is repeated for the other macroblocks in the frame. These values were obtained considering only the respective core algorithms and do not take into account the impact of communication or operating system.

TABLE III. COMPARING CYCLE-ACCURATE SIMULATION AND ESTIMATION

Benchmark	Cycle-accurate execution time	Estimation time	Estimation error (%)
Matrix sum	9 sec	0.39 sec	3%
LMS filter	12 sec	0.52 sec	1%
MPEG Encoder	600 sec	17 sec	17%

If we consider a frame of 174x144 pixels, the encoder will divide it in 99 macroblocks. From our estimation tool, the macroblock execution cost is 122,910 cycles, and consequently a complete frame encoding will demand

12,168,090 cycles. Using a rate of 25 frames/second, the deadline for frame processing is 40 milliseconds. The real-time requirement could be respected using an Encoder processor running at 265 MHz. On the other hand, if the processor frequency is fixed at 100 MHz, this will require 3 processors executing the encoder task in parallel.

C. *Virtual prototype simulation*

To validate our concept, we use an architecture composed of one processor to execute the Encoder and another one to execute the VLC task. After the HW/SW interface refinement, we use a virtual prototype to obtain performance results using simulation. This integrated design and performance analysis approach eases the virtual prototype generation and evaluation.

To evaluate the performance of the refined design, a virtual prototype was produced using the MaxSim environment. In this work, a tool automatically imports our design to MaxSim. It is responsible for the encapsulation of the SystemC IP components as MaxSim components. We consider that the supplier provides hardware IP components as cycle-accurate models. The design hierarchy is preserved in the virtual prototype, thus easing the system analysis.

The global simulation model provides a suitable way to evaluate hardware and software components. Analyzing the exact software execution time in MaxSim, and comparing it with results obtained by the neural network estimator, we obtain an estimation error of 10.28% for the Encoder task and 17% for the VLC task. These errors have two sources. The first source is the intrinsic error related to the neural network estimation. The second one is related to the overhead of the operating system and drivers, which are not considered in the dynamic instruction count used as input to the neural network. In fact, at specification level we are exploring the architecture, and consequently the operating system and the communication API are not yet defined.

An important remark is that the communication between the processors is implemented by the DMA component and thus does not require shared resources such as buses and memories. The estimator is trained considering a monoprocessor system without concurrent accesses to the memories. In the case of an SMP architecture, load/store costs may have a high variability and the estimation could result in a larger error.

VII. CONCLUSIONS

Early performance estimation and analysis tools have recently attracted the attention of the research community due to the complexity and heterogeneity of the current and future embedded systems. Fast and accurate performance estimation tools are needed to help the design architecture exploration.

This work proposes an integrated methodology for design and performance analysis. Processor selection is supported by a software performance estimator based on neural networks. The estimator provides flexibility and precision even for complex processors, with pipeline and cache memories. The estimator is fast compared to the cycle-accurate simulation, as presented in Table III, helping the architecture exploration by enabling a rapid processor selection.

After the processor selection and architecture exploration, a “golden” virtual architecture is created. This architecture is then refined to a BFM model containing processors, IP blocks, memories, and peripherals. The software part is composed of software tasks running under an operating system. The simulation model for system evaluation is automatically generated from the architecture description in the MaxSim environment. This integration allows the designer to spend time in the design analysis, and not in the virtual prototype modeling.

Future work will investigate the use of a virtual architecture using TLM channels to estimate the communication requirements. The estimation at this level will help in design decisions regarding the implementation of HW and SW interfaces. The OS overhead is considered only in the virtual prototype simulation, but not in the neural network estimator. A possible extension is to take into account the system calls in the neural network model, with a fixed cost, thus increasing the estimation precision.

REFERENCES

[1] M.S.OYAMADA, F.ZSCHORNACK, F.R.WAGNER. “Accurate Software Performance Estimation Using Domain Classification and Networks”. In: 17th Symposium on Integrated Circuits and Systems Design, September 2004.

[2] W.O. CESARIO, D. LYONNARD, G. NICOLESCU, Y. PAVIOT, S. YOO, L. GAUTHIER, M. DIAZ-NAVA, A.A. JERRAYA, “Multiprocessor SoC Platforms: A Component-Based Design Approach”, IEEE Design & Test of Computers, Vol. 19, Nr. 6, Nov-Dec, 2002.

[3] M.LAJOLO, M.LAZARESCU, A. SANGIOVANNI-VINCENTELLI. “A Compilation Based Software Estimation Scheme for Hardware/Software Co-simulation”. In: Symposium on Hardware/Software Codesign, 1999.

[4] P.GIUSTO, G.MARTIN, E.HARCOURT. “Reliable Estimation of Execution Time of Embedded Software”. In: Design, Automation and Test in Europe, March 2001.

[5] G.BONTEMPI, W.KRUIJTZER. “A Data Analysis Method for Software Performance Prediction”. In: Design, Automation and Test in Europe, March 2002.

[6] J.BAMMI et al. “Software Performance Estimation Strategies in a System-level Design Tool”. In: Symposium on Hardware/Software Codesign, May 2000.

[7] ARM MaxSim. http://www.arm.com

[8] ConvergenSC. http://www.coware.com

[9] Synopsys System Studio. http://www.synopsys.com

[10] L.BENINI, D.BERTOZI, A.BROGIOLO, F.MENICHELLI, M.OLIVIERI. “MPARM: Exploring the Multi-Processor SoC Design Space with SystemC”. In: Journal of VLSI Signal Processing, Vol. 41, 2005.

[11] A.WIEFERINK et al. “A System Level Processor/Communication co-exploration Methodology for Multi-processor System-on-chip Platforms”. In: Design, Automation and Test in Europe, February 2004.

[12] J.FREEMAN, D.SKAPURA. “Neural Networks: Algorithms, Applications and Programming Techniques”. Addison-Wesley Publisher, 1992.

[13] MATLAB – Neural networks toolbox. http://www.mathworks.com

[14] J.ENGBLOM, A.ERMEDAHL, F.STAPPERT. “A Worst-Case Execution-Time Analysis Tool Prototype for Embedded Real-Time Systems”. In: Workshop on Real-Time Tools, 2001

[15] Y.-T.S.LI, S.MALIK. “Performance Analysis of Embedded Software Using Implicit Path Enumeration”. In: Design Automation Conference, June 1995.

[16] M.BONACIU et al. "High-Level Architecture Exploration for MPEG4 Encoder with Custom Parameters". In: 11th Asia and South Pacific Design Automation Conference, January 2006.