System-Level Power/Performance Analysis for Embedded Systems Design†

Amit Nandi Radu Marculescu

Department of Electrical and Computer Engineering
Carnegie Mellon University,
Pittsburgh, PA 15213, U.S.A.

{anandi,radum}@ece.cmu.edu

Abstract: *This paper presents a formal technique for system-level power/performance analysis that can help the designer to select the right platform starting from a set of target applications. By platform we mean a family of heterogeneous architectures that satisfy a set of architectural constraints imposed to allow re-use of hardware and software components. More precisely, we introduce the Stochastic Automata Networks (SANs) as an effective formalism for average-case analysis that can be used early in the design cycle to identify the best power/performance figure among several application-architecture combinations. This information not only helps avoid lengthy profiling simulations, but also enables efficient mappings of the applications onto the chosen platform. We illustrate the features of our technique through the design of an MPEG-2 video decoder application.*

Keywords: platform-based design, system-level analysis, stochastic automata networks, multimedia systems.

1. Introduction and objectives

This paper presents a technique for *system-level power/performance analysis* that can be used in platform-based design [1,2]. By *platform* we mean a family of heterogeneous architectures that satisfy a set of architectural constraints imposed to allow re-use of hardware and software components. While the technique that we propose is completely general and therefore can be used with any embedded application, we focus our presentation on *portable embedded multimedia systems*. These systems are characterized by "*soft*" real-time constraints, and hence, as opposed to safety critical systems, their *average* behavior is far more important than the worst-case behavior. Indeed, due to data dependencies, their computational requirements show such a large spectrum of statistical variations that designing them based on the worst-case behavior (typically, orders of magnitude larger than the actual execution time [3]) would result in completely inefficient systems.

†Research supported in part by NSF under grant CCR-00-93104 and in part by a grant from Intel Corporation.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

Typically, the design of heterogeneous architectures, follows the Y-chart scheme [4]. In this scheme, the designer first characterizes the set of target applications and chooses a family of candidate architectures to run that set. Then, the application is mapped onto the architectural components and the performance of the system is evaluated. Based on the resulting performance numbers, one may decide to go ahead with the chosen architecture. Otherwise, if the performance figures are not satisfactory, the designer may restructure the application, or modify the mapping of the application to get better performance numbers. Relying upon this Y-chart design methodology, we focus on the *application-architecture modeling* process for embedded multimedia systems.

Our global vision is presented in Fig.1. Following the principle of orthogonalization of concerns during the design process [2], we build *separate* models for applications and architectures. Next, we *map* the abstract model of an application onto a family of architectures (platform) and *evaluate* the power/performance figures to see how suited is the platform (and the chosen set of design parameters) for the target application. This process can be re-iterated with a different set of parameters until convergence.

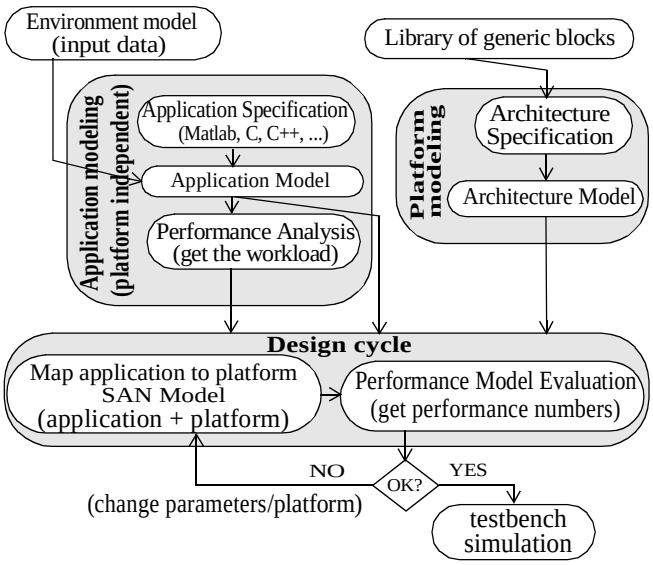

Fig. 1: Our vision for system-level performance analysis

Our vision has several unique features: First, our methodology is based on integrating the power/performance metrics into system-level design. Indeed, the performance metrics that we develop in the application and platform modeling step become an integral part

of the design process; this helps the system designer to quickly find the right architecture for the target application. Second, using the same unique representation based on *Stochastic Automata Networks* SANs (for both application and architecture) gives the ability to smoothly translate a performance model into an architecture (obtain real numbers on performance), and reflect architectural changes back to the performance model.

1.1. Contribution of the paper

The key contribution of this paper is the new idea of using SANs [5,18] as an effective formalism in system-level analysis. SANs are a very powerful Markovian formalism belonging to the class of process algebras which are very efficient in modeling communicating concurrent processes. A major advantage of SANs over other formalisms is that the state space explosion problem associated with the Markov models (or Petri nets) is partially mitigated by the fact that the state transition matrix is *not* stored, nor even generated.

The models that we build for applications are *process-level* functional models that are free of any architectural details. The processes communicate and interact among them defining what the application should do and not how it will be implemented. On the other hand, the architecture models represent behavioral descriptions of the architectural building blocks. Typically, these building blocks may consist of several programmable cores or dedicated hardware units, communication resources (buses) and memory resources (RAMs, FIFO buffers). A separation of concerns between application and architecture, enables reuse of both application and architecture models and facilitates an explorative design process in which application models are subsequently mapped onto architecture models.

Once built, the application-architecture model is evaluated to analyze the characteristics of the processes for different input parameters. While model evaluation is a challenging problem by itself, *analytical* performance model evaluation presents additional challenges. No other proposed evaluation strategy for platform-based design supports analytical calculations for communicating and interacting processes that represent multimedia applications. To this end, we develop a fully analytical framework using SANs to avoid lengthy simulations for predicting power and performance figures. This is important for multimedia systems where thousands of runs are typically required to gather relevant statistics for average-case behavior. Considering that 5 min. of compressed MPEG-2 video needs roughly 1.2 Gbits of input vectors to simulate, the impact of having such a tool to evaluate power/performance estimates becomes evident.

1.2. Related work

Most of the research in performance analysis was geared so far towards the *worst-case analysis*, where the correctness of the system depends not only on the logical results of computation, but also on the time at which the results are produced [7,8]. Despite the great potential for embedded system design, the area of average-case analysis received little attention [3,9,10]. The target of our research is to investigate this very issue and, using abstract representations, provide quantitative measures of power/performance estimates. Our effort *complements* the existing results for worst-case time analysis and is distinct from other approaches for performance analysis based rate analysis [11], and adaptation process [3]. Compared to our approach, none of these approaches handles applications at process-level using *communicating* and *interacting* processes and yet provides performance metrics that can be used in platform-based design. We note that existing tools for high-level performance modeling that can be used in embedded systems design, like Ptolemy [12] and Polis[13], use simulation for performance evaluation.

In summary, we propose a completely analytic solution for application-architecture modeling for performance evaluation of networked multimedia systems. What makes this unique is the potential to significantly shorten the design cycle, while still providing the ability to explore thoroughly the design space.

1.3. Organization of the paper

Section 2, presents some background information on SAN modeling paradigm. In Section 3, we present the details of our modeling technique and show how this can be used in practice to analyze the MPEG-2 video decoder. In Section 4 we discuss the power/performance results for several scenarios and illustrate the possible implications of the analysis results in the design process. Finally, we conclude by summarizing our main contribution.

2. The SAN modeling Paradigm

SANs present a modular state-transition representation for highly concurrent systems. The main objective of SAN analysis is the computation of the stationary probability distribution π for an *N*-dimensional system consisting of *N* stochastic automata that operate more or less independently. This involves two major steps: 1) SAN model construction and 2) SAN model evaluation. The following sections briefly describe these two steps. For more details, the reader is referred to [5].

2.1 The SAN model construction

The SAN model can be described using continuous-time Markov processes which are based on *infinitesimal generators*:

$$Q = \begin{bmatrix} -\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \cdots \\ \sigma_{1,0} & -\sigma_{1,1} & \sigma_{1,2} & \cdots \\ \sigma_{2,0} & \sigma_{2,1} & -\sigma_{2,2} & \cdots \end{bmatrix} \quad (1)$$

with $\sigma_{i,i} = \lim_{t \to 0} \frac{1 - p_{i \to i}}{t} = -p'_{i \to i} \quad i = 1, 2, \ldots, n$,

$\sigma_{i,j} = \lim_{t \to 0} \frac{p_{i \to j}}{t} = p'_{i \to j} \quad i, j = 1, 2, \ldots, n \ (i \neq j)$, and

$\sum \sigma_{i,j} = \sigma_{i,i} \quad i, j = 1, 2, \ldots, n (i \neq j)$, where $p_{i \to j}$ is the transition probability from state i to state j during time 0 to t, and $p'_{i \to j}$ is its derivative. Each entry σ_{ij} in the infinitesimal generator is the *execution rate* of the process in that particular state [3,11].

There are two ways in which the automata (processes) may interact:

- A transition in one automaton may force a transition to occur in one or more other automata. These are called *synchronizing* transitions (events). They affect the global system by altering the state of possibly *many* automata. In any given automaton, transitions that are not synchronizing transitions are said to be *local* transitions.

• The *rate* at which a transition may occur in one automaton may be a function of the state of other automata. These transitions that depend on other external conditions are called *functional* transitions as opposed to *constant-rate* (*non-functional*) transitions.

Given N stochastic automata (with associated infinitesimal generators $Q^{(1)}, Q^{(2)}, \ldots, Q^{(N)}$) that interact via E synchronizing events (and no functional transition rates), the infinitesimal generator of the system can be written as:

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{i=1}^{N} Q_j^{(i)} \qquad (3)$$

This quantity is also called the *global descriptor* of the SAN and it is written as a sum of tensorial[1] products. On introducing functional transition rates this descriptor can still be written as in eqn. (3), but now the elements of $Q_j^{(i)}$ may be functions. Thus eqn. (3) becomes

$Q = \sum_{j=1}^{T} \bigotimes_{i=1}^{N} \overline{Q}_j^{(i)}$, where $\overline{Q}$ contains only numerical values and the size of T depends on $(2E+N)$ and on $\prod_{i \in F} n_i$, where F is the set of automata whose state variables are arguments in functional transition rates.

2.2 Performance model evaluation

Once we have the SAN model, we need to find out its *steady-state* behavior. This is simply expressed by the solution of the equation

$$\pi \cdot Q = 0 \qquad (4)$$

with the normalization condition $\pi \cdot e = 1$, where π is steady-state probability distribution and $e^T = (1,1,...,1)$.

We solve eqn. (4) using numerical methods that *do not* require the explicit construction of the matrix Q but can work with the descriptor in its compact form (namely, *iterative methods*). This can be done using $\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i$ multiplications, where n_i is the numbers of states in the *i-th* automaton [5,6].

Once the steady-state distributions are known, performance measures such as *throughput, utilization, average response time* can be easily derived. However, to calculate these performance figures, we need to find the *true rates* of the activities. This is because the specified rate of an activity is *not* necessarily the rate of that activity in the equilibrium state, since bottlenecks elsewhere in the system may slow the activity down. Thus, the true (or equilibrium) rate of an activity can be obtained by multiplying the given rate with the *probability* of the activity being enabled.

1. $X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$, $Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$, $X \otimes Y = \begin{bmatrix} x_{11}Y & x_{12}Y \\ x_{21}Y & x_{22}Y \end{bmatrix}$

3. Modeling the MPEG-2 Video Decoder Application

In what follows, we describe the main steps in Fig.1 using the MPEG-2 video decoder as the driver application.

3.1 System specification

The decoder consists of the baseline unit, the Motion Compensation (MV) unit, recovery unit, and the associated buffers (Fig.2). The baseline unit contains the VLD (Variable Length Decoder), IQ/IZZ (Inverse Quantization and Inverse Zigzag), the IDCT (Inverse Discrete Cosine Transform) and the buffer. During the modeling steps, we model each of these units as processes, and generate their corresponding SANs.

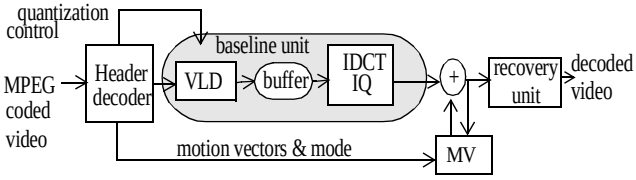

Fig.2 The block diagram of the MPEG-2 decoder

To specify our system, we chose the Stateflow component of Matlab which uses the semantics of Statecharts [14]. Statecharts extend the conventional state diagrams with the notion of hierarchy, concurrency and communication. This is important since we aim to analyze how the asynchronous nature of concurrent systems can affect their run-time behavior.

3.2 Application Modeling

To model the applications of interest, we use a *process graph*, where each component corresponds to a process in the application. Communication between processes is achieved using *event* and *wait* synchronization signals. Process graphs are also characterized by *execution rates* which, under the hypothesis of exponentially distributed sojourn times, can be used to generate the underlying Markov chain [6]. In our SAN-based modeling strategy, each automaton corresponds to a process in the application. Hence, the whole process graph specifying the embedded system translates to a *network of automata*.

We model the entire process graph that corresponds to MPEG-2 following the *Producer-Consumer* paradigm [15]. To unravel the complete concurrency of processes that describe the application, we assume that each process has its own space to run and does not compete for any computing resource. For the sake of simplicity, we present in Fig.3, the SAN model just for the baseline unit.

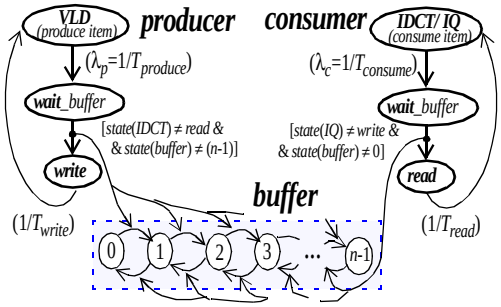

Fig.3 The SAN model of the baseline unit of MPEG-2

Referring to the *Producer* process (*VLD*), we observe a local transition between *produce*(*item*) and *wait_buffer* states; that is, this transition occurs at the fixed rate of $1/T_{produce}$ where $T_{produce}$ is the required time to produce one item. The transition from the state *wait_buffer* to the state *write* is a functional transition because it depends on the state of the other process. More precisely, this transition happens if and only if the process *IDCT* is not reading any data and the buffer is not full. Because of this dependency, we cannot associate a fixed rate to this transition; the actual rate will depend on the overall behavior of the system. Finally, once the producer gets access to the buffer, it transitions to the initial state (the local transition rate is $1/T_{write}$). The same considerations apply to the *Consumer* process (*IDCT/IQ*).

3.3 Architecture modeling

This modeling step starts with an abstract specification of the platform (e.g. Stateflow) and produces a SAN model that reflects the behavior of that particular specification. We construct a *library* of generic blocks that can be combined in a bottom-up fashion to model sophisticated behaviors. The generic building blocks model different types of resources in an architecture, such as processors, communication resources, and memory resources. Defining a complex architecture thus becomes as easy as instantiating building blocks from a library and interconnecting them. Compared to the laborious work of writing fully functional architecture models (in Verilog/VHDL), this can save the designer a significant amount of time, and therefore enable exploration of alternative architectures.

Architecture modeling shares many ideas with application modeling that was just discussed. Without further details, we illustrate in Fig.4 a few simple generic building blocks.

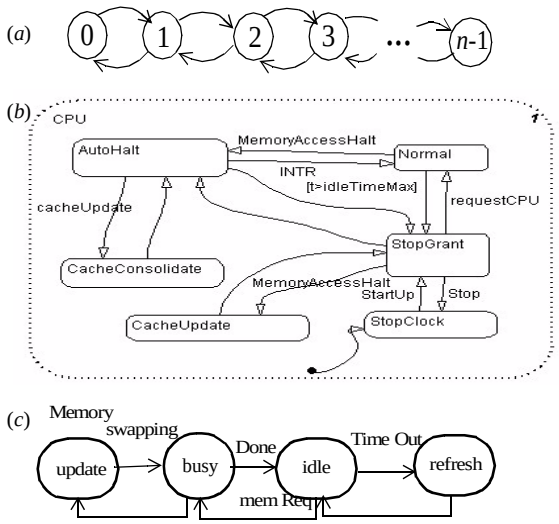

Fig.4 The SAN models of some generic blocks

In Fig.4a, we represent a buffer of max length (*n*-1), where state 0 corresponds to the empty buffer, while state (*n*-1) to the full buffer. Every time when a request for a new insertion occurs, the current state of buffer changes with one position to the right. Similarly, every time when there is a request to delete an item from the buffer, a transition from the current state of the buffer to the left position occurs. In Fig.4b, we have the generic model of a CPU based on a power saving architecture. *Normal*-mode is the normal operating mode when every on-chip resource is functional. *StopClock* mode offers the greatest power savings and consequently the least functionality. Finally, Fig.4c describes a typical memory model.

These simple examples were presented for illustrative purposes. We have no limitation whatsoever, to build much more elaborated models and use them in real life examples.

3.4 Mapping

Having the application and the architecture models, the next step is to *map* the application onto architecture and then *evaluate* the model using the analytical procedure in Section 2. To make the discussion more specific, let us consider the following design problem.

The design problem

Assume that we have to decide how to configure a platform which can work in four different ways: one which has three identical CPUs operating at a generic clock frequency f_0 (then each process can run on its own processor) and another three architectures where we can use only one physical CPU, but have the freedom of choosing its speed among the values f_0, $2f_0$, or $3f_0$.

Mapping our simple *VLD-IDCT/IQ* processes in Fig.3 onto a platform with a single CPU is illustrated in Fig.5. Because these processes[1] have to share now the same CPU, some of the local transitions become synchronizing/functional transitions (e.g. the local transitions with rates $1/T_{produce}$ or $1/T_{consume}$ become synchronized). Moreover, some new states (e.g. *wait_CPU*) have to be introduced to model the new synchronization relationship.

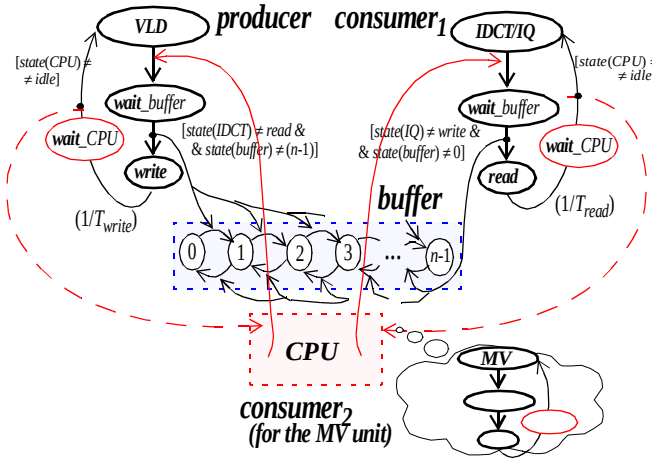

Fig.5 Application mapping onto a single CPU platform

To complete the mapping, we need another process, namely the *scheduler*. This process determines the sequence in which the various concurrent processes of the application run on the different architectural components, particularly if the resource is shared[2]. This process can be implemented in software or hardware but since our SAN representation is uncommitted, we can easily add this new component to the entire network of automata. This completes all steps of our the modeling methodology.

1. For simplicity, the second consumer process (for the MV unit) was not explicitly represented in this figure.
2. Our scheduler has now a simple FCFS policy but we plan to incorporate other scheduling policies, in the near future.

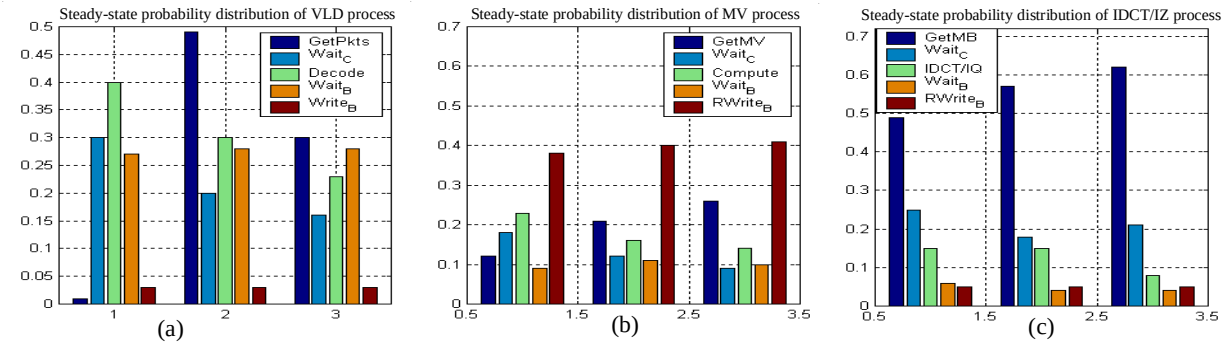

Fig.6 Steady-state probability distributions of *(a)*VLD, *(b)*MV and *(c)*IDCT/IQ unit

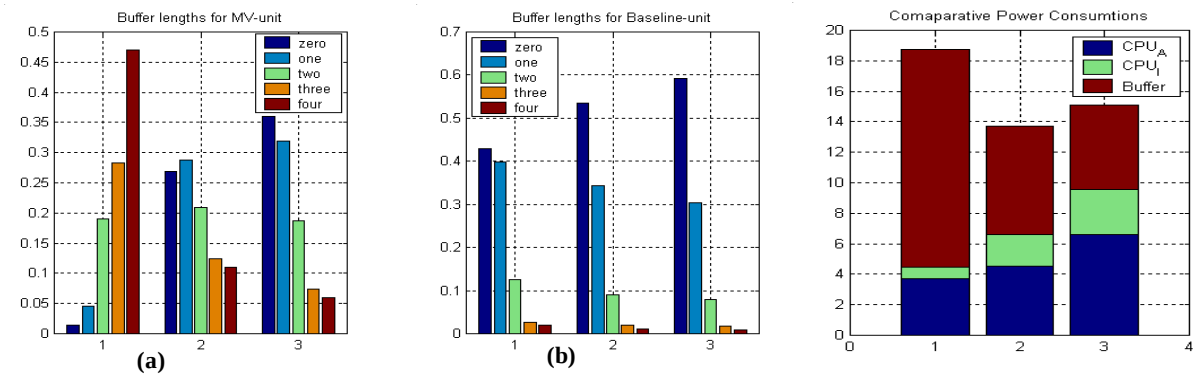

Fig.7 Buffer Length for the (a) MV (b)IDCT/IQ -unit for f_0, $2f_0$, and $3f_0$

Fig.8 Comparative Power-consumption figures for f_0, $2f_0$, and $3f_0$

4. Results and discussion

The input value for T_{VLD} is computed from the MPEG-2 application requirements and characteristics of the input traces. More precisely, for NTSC video, we have (720×480)[pixels]×30[frames/sec.]; that is, the number of macroblocks/sec is (45×30)×30=40500, which means 24 µsec necessary to produce one macroblock. For all the runs, we use the following parameters: n = 5 (one entry in the buffer represents one block of 64 DCT coefficients that are needed for one IDCT operation), T_{VLD} = ($T_{produce}$) = 20 µsec, T_{IDCT} = ($T_{consumer1}$) = 12 µsec, T_{MV} = ($T_{consumer2}$) = 14 µsec, T_{write} = 5 µsec and T_{read} = 12 µsec.

4.1 Performance results

For the case of having a platform with three separate CPUs, the analysis is quite simple: the system will essentially run in the CPU-active state or will either be waiting for the buffer or writing into the buffer most of the time. The average length values are 1.57 and 0.53 for the MV and baseline unit buffers, respectively. This is in deep contrast with the worst-case scenario assumption where the lengths will be 4 across all runs.

In the case of having a platform with a single CPU, the probability distribution values for all the components of the system are given in Fig.6. The first column in these diagrams shows the probability of the processes waiting for their respective packets to arrive. The second column shows the probability of the process waiting for the CPU to become available (because the CPU is shared among all three processes). The third column represents the probability of the processes actively using the CPU to accomplish their specific tasks. The fourth column shows the probability of the process being blocked because the buffer is not available (either it is full/empty or used by another process). The fifth column shows the probability of the processes writing into their corresponding buffers.

Run 1 represents the 'reference' case where the CPU operates at frequency f_0, while the second and the third runs represent the cases when the CPU speed is $2f_0$ and $3f_0$, respectively. For instance, in run 1, the Producer (VLD) is waiting with probability 0.01 to get its packets, waiting for CPU access with probability 0.3, decoding with probability 0.4, waiting for the buffer with probability 0.27, and finally writing the data into the buffer with probability 0.02.

Looking at the probability distribution values of MV and baseline unit buffers[1] (Fig.7), we see that the bottleneck may appear because of the MV buffer. More precisely, the system is overloaded at run1, balanced at run 2 and under-utilized at run 3. The average buffer lengths in runs 1, 2 and 3 are: MV buffer: 3.14, 1.52, and 1.15; baseline unit buffer: 0.81, 0.63, and 0.54, respectively. Since the average length of the buffers is proportional to the average waiting time (and therefore directly impacts the system performance), we can see that, based solely on performance figure, the best choice would be a single CPU with speed $3f_0$. Also, we notice how different the average values (e.g. 1.15 and 0.54, respectively) are compared to the value 4 provided by a worst-case analysis. Not only this worst-case length is about 4 times larger than the average one, but it also occurs in less that 6% of the time. Consequently, design-

1.The columns in the Buffer diagrams show the distribution of the buffer occupancy ranging from 0 (empty) to 4 (full).

ing the system based on a worst-case analysis will result in a completely inefficient implementation.

4.2 Power results

The *average system-level power* can be obtained by summing up all the subsystem-level power values. For any subsystem *k*, the average power consumed is given by:

$$P^{(k)} = \sum_{all\ i} \pi_i \cdot P_i + \sum_{all\ i,j} \lambda_{ij} \cdot P_{ij} \qquad (5)$$

where P_i and P_{ij} represent the power consumption per state and per transition, respectively, and π_i is the steady-state probability and λ_{ij} is the transition rate associated with the transitions between states *i* and *j*. Having already determined the solution of eqn. (4), the π_i value (for a particular *i*) can be found by summing up the appropriate components of the global probability vector π. The P_i and P_{ij} costs are determined during an off-line pre-characterization step where other proposed techniques can be successfully applied [16].

To obtain the power values, we simulated the MPEG-2 decoder, using the Wattch [17] architectural simulator that estimates the CPU power consumption, based on a suite of parametrized power modes. More precisely, we monitored the simulation of the MPEG-2 and extracted the power values for all components of the system. By specifying a Strong-Arm-like processor, we obtained an average power value of 4.6W for the VLD, and 4.8W for the IDCT and 5.1W for the MV unit. Using these the power figures, we obtained the average power characterization for the entire system under varying loads. This is useful to trade-off performance and power. In our case, using eqn. (5), we have the (total) average power values of 18.75W, 13.68W, 15.08W for runs 1, 2, and 3, respectively.

For a more detailed analysis, the breakdown of power-consumption is given in Fig.8. We can see that there is a large variation among the three runs with respect to both the CPU-active power, and the power dissipation of the buffers. Furthermore, we can multiply these power values with the average buffer lengths from Fig.7 (3.14, 1.52, and 1.15, for runs 1, 2 and 3, respectively), and get the *power×delay* characterization of the system; that is, 58.9 [μJ/macroblock], 20.8 [μJ/macroblock], and 17.3 [μJ/macroblock] (about 70% less) for runs 1, 2, and 3, respectively. This analysis shows that the best choice would be to use the third configuration (e.g. CPU running at $3f_0$) since, for the given set of parameters, it represents the best application-architecture combination. (This choice is also far better than using three separate CPUs all running at speed f_0.)

Finally, the CPU time needed for our analysis is several orders of magnitude better that the active simulation time required to obtain the same results with blind simulation. Hence, the approach can significantly cut down the design cycle time and, at the same time, enhance the opportunities for better exploration of the design space.

5. Conclusion

We have presented a formal technique for system-level analysis based on SANs. The proposed methodology *complements* the existing techniques for extreme-case performance analysis by incorporating the environment characteristics into system performance evaluation. Being targeted at system-level, the results of this analysis are not confined to any particular hardware/software implementation so they can offer the generality and flexibility designers need in designing the embedded system. Experimental results have been presented for an MPEG-2 video decoder showing several orders of magnitude speedup compared to explicit simulation.

6. Acknowledgements

The authors would like to thank Alberto Sangiovanni-Vincentelli, Luciano Lavagno, and Diana Marculescu, for many stimulating discussions on this topic.

7. References

[1] S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vincentelli, 'Design of embedded systems: formal models, validation, and synthesis,' *Proc.IEEE*, Vol.85, no.3, March, 1997.

[2] M. Sgroi, L. Lavagno, A. Sangiovanni-Vincentelli, 'Formal Models for Embedded Systems Design,' *IEEE Design and Test of Computers*, Vol. 17, April-June 2000.

[3] A. Kalavade, P. Moghe, 'A tool for performance estimation of networked Embedded End-Systems,' *Proc. DAC*, San Francisco, CA, June 1998.

[4] P. Lieverse, P. Van der Wolf, E. Deprettere, K. Vissers, 'A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems,' *Proc. Workshop in SiPS*, Taipei, Taiwan, 1999.

[5] B. Plateau, J. M. Fourneau, 'A Methodology for Solving Markov Models of Parallel Systems,' *Journal of Parallel and Distributed Camp.*, Vol. 12, 1991.

[6] W. J. Stewart, 'An introduction to the Numerical Solution of Markov Chains,' Princeton Univ. Press, New Jersey, 1994.

[7] S. Malik, M. Martonosi, Y.-T. Li, 'Static Timing Analysis of Embedded Software,' *Proc. DAC*, Anaheim, CA, 1997.

[8] T.-Y. Yen, W. Wolfe, 'Performance Estimation for Real-time Distributed Embedded Systems,' *Proc. ICCD*, 1995.

[9] Y. Shin, D. Kim, K. Choi, 'Schedulability-Driven Performance Analysis of Multiple Mode Embedded Real-Time Systems,' *Proc. DAC*, Los Angeles, CA, June 2000.

[10] T. Zhou, X. Hu, E. Sha, 'A Probabilistic Performance Metric for Real-Time System Design,' *Proc. CODES*, May 1999.

[11] A. Mathur, A. Dasdan, R. Gupta, 'Rate Analysis for Embedded Systems,' *ACM Trans. on Design Automation of Electronic Systems*, Vol. 3, no. 3, July 1998.

[12] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, 'Ptolemy: A framework for simulating and prototyping heterogeneous systems,' *Intl. Journal of Computer Simulation*, Vol. 4, Apr. 1994.

[13] F. Balarin et. al., 'Hardware-Software Co-design of Embedded Systems - The POLIS approach', Kluwer Academic Publishers, 1997.

[14] D. Harel, 'Statecharts: A visual formalism for complex systems,' in *Sci. Comp. Prog*, Vol. 8, 1987.

[15] R. Marculescu, A. Nandi, 'Probabilistic Application Modeling for System-Level Performance Analysis', *Proc. DATE*, Munich, Germany, March, 2001.

[16] T. Simunic, L. Benini, G. De Micheli, 'Cycle-Accurate Simulation of Energy Consumption in Embedded Systems,' *Proc. DAC*, New Orleans, June 1999.

[17] David Brooks, Vivek Tiwari, Margaret Martonosi, 'Wattch: a framework for architectural-level power analysis and optimizations,' *Proc. ISCA*, June 2000.

[18] B. Plateau, K. Atif, 'Stochastic Automata Network for Modelling Parallel Systems,' IEEE Trans. on Software Engineering, Vol. 17, Oct. 1991.