

# Probabilistic Application Modeling for System-Level Performance Analysis

Radu Marculescu

Amit Nandi

Department of Electrical and Computer Engineering  
Carnegie Mellon University, Pittsburgh, PA 15213 USA

**Abstract:** *The objective of this paper is to introduce the Stochastic Automata Networks (SANs) as an effective formalism for application modeling in system-level analysis. More precisely, we present a methodology for application modeling for system-level power/performance analysis that can help the designer to select the right platform and implement a set of target multimedia applications. We also show that, under various input traces, the steady-state behavior of the application itself is characterized by very different ‘clusterings’ of the probability distributions. Having this information available, not only helps to avoid lengthy profiling simulations for predicting power and performance figures, but also enables efficient mappings of the applications onto a chosen platform. We illustrate the benefits of our methodology using the MPEG-2 video decoder as the driver application.*

**Keywords:** system-level design, performance analysis, application modeling, stochastic automata networks, embedded multimedia systems.

## 1. Introduction and objectives

Embedded systems represent an important segment of today’s electronic industry. While there has been a notable growth in the use and application of these systems, the design process has become an increasingly difficult problem due to the increasing design complexity and shortening time-to-market [1-4]. Practical optimization is impossible without (1) efficient, yet accurate *performance models* at higher levels of abstraction, and (2) *tools* that let us quickly evaluate proposed changes and their impact at the highest level of abstraction, alleviating the need to defer to a fully detailed implementation of the system.

The objective of this paper is to present a technique for *application modeling* 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 understand a family of *heterogeneous* architectures (consisting of both programmable and dedicated components) that satisfy a set of architectural constraints imposed to allow reuse of hardware and software components [5].

While the technique that we propose is completely general, the focus of our presentation is on *portable embedded multimedia systems*. These systems are characterized by “*soft*” real-time constraints and then, as opposed to reactive embedded systems used in safety critical applications, the 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, one or two orders of magnitude larger than the actual execution time [6]) would result in completely inefficient systems. Also, since the *Quality of Service* (QoS) requirements can vary considerably from one media type to another (e.g. video connections require consistently high throughput, but tolerate reasonable levels of jitter and bit or packet errors), the ability to explore several design alternatives while trying to satisfy QoS requirements is of crucial importance. To this end, estimating the power/performance figures of a *set* of applications, with respect to a target platform, is one of the key problems that needs to be addressed.

Typically, the design process using heterogeneous architectures follows the Y-chart scheme (Fig.1) [7]. In this scheme, the designer first characterizes the set of applications and chooses an architecture to run that set. Then, each application is mapped onto the architecture and the performance of each application-architecture mapping is evaluated. Depending on the resulting performance numbers, the designer may decide to use that architecture, restructure the application, or modify the mapping of the application to get better performance numbers.

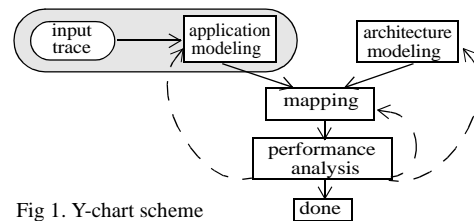


Fig 1. Y-chart scheme

Relying upon this Y-chart design methodology, we focus on the *application modeling* step for embedded multimedia systems (shaded area in Fig.1). This is motivated by our observation that, constraining a given application (e.g. MPEG-2) with various input traces (e.g. MPEG-coded video movies with very different scene changes) leads to very different ‘clusterings’ of the probability distribution that characterize the application itself. Tuning the target architecture to this large spectrum of different probability distributions is the most important development in obtaining efficient mappings with respect to certain performance metrics.

### 1.1. Contribution of the paper

The key contribution of this paper is the new idea of using *Stochastic Automata Networks* (SANs) [8][9] as an effective formalism for application modeling in system-level analysis. SANs are powerful Markovian formalism belonging to the class of process algebras which are very efficient in modeling communicating processes. This property is useful since embedded applications are highly concurrent and conse-

quently, do not easily fit the traditional model of sequential control flow. Another advantage of SANs over other formalisms is that the state space explosion problem associated with the Markov models (or Petri nets) is 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 to specify what the application should do, not how it will be implemented. Once the model is built, the next step is model evaluation. 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. Based on SANs, we develop a fully analytical framework that can help to avoid lengthy profiling simulations for predicting power and performance figures. This is important for multimedia systems since 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.

Taken together, our proposed technique allows media systems designers to explore architectures more rapidly, estimate the impact of different design choices more robustly, and use large multimedia data benchmarks more effectively.

### 1.2. Related work

Embedded systems interact with the outside world and, in many cases, their interactions have to satisfy strict timing constraints. This characteristic drove most of the research towards the *worst-case analysis* which means that 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 [10-12]. Despite the great potential for embedded system design, the area of average-case analysis received little attention [6][14]. 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 quite distinct from other approaches for performance analysis based on time separation between events [15], rate analysis [16], and adaptation process [6]. 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.

Another important issue is to relate our solution to other existing tools for high-level performance modeling that can be used in embedded systems design. Ptolemy [17] focuses on application modeling and simulation, but does not yet support explicit mapping of application models onto models of architectures. The Polis environment [18] is very well-suited for reactive systems, but less suited for applications involving

DSP kernels. Chinook [19] supports an explicit mapping from a behavioral description onto a target architecture and again, a simulation tool is used to simulate the system at different levels of abstraction. El Greco [20] provides a simulation environment for modeling and validating the functionality of complex heterogeneous systems. Finally, the tools recently proposed in [7] are centered around the idea of platform-based design. The applications are modelled as Kahn process networks that are further used to perform performance evaluation via simulation.

In summary, we propose a completely analytic solution for application modeling for performance evaluation. What makes this unique is the potential to significantly shorten the design cycle, while providing designers the ability to explore the entire design space. Indeed, by its very nature, our analytic approach is fast and scales nicely with the design complexity.

### 1.3. Organization of the paper

Section 2, presents the SAN modeling paradigm. In Section 3, we present a detailed analysis of application modeling for the MPEG-2 decoder and discuss the power/performance results for two different scenarios. In Section 4, we illustrate possible implications of the technique on the design process. Finally, we conclude by summarizing our main contribution.

## 2. Application modeling using SANs

To model the application of interest, we use a *process graph*, where each node (component) corresponds to a process in the application. More precisely, we associate an automaton to each process in the application and hence the whole process graph specifying the application translates to a *network of automata*. Furthermore, the process graph is characterized by *execution rates* which, under the hypothesis of exponentially distributed activity durations, can be used to generate the underlying Markov chain [9]. We also note that, in our SAN-based modeling strategy, the processes are fully concurrent and the synchronization between them occurs according to a statically defined relation. Communication between processes can be achieved using various protocols, including simple ones based on *event* and *wait* synchronization signals.

### 2.1 The SAN model construction

One of the central concepts in the theory of continuous Markov processes is the *infinitesimal generator*<sup>1</sup> defined as:

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

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

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

<sup>1</sup>. This generator is the analogue of the transition probability matrix in the discrete-time Markov chain.

$$\sum_{i \neq j} \sigma_{i,j} = \sigma_{i,i} \quad i, j = 1, 2, \dots, n \quad (i \neq j),$$

where  $p_{i \rightarrow j}$  is the transition probability (directly or indirectly) from state  $i$  to state  $j$  during time 0 to  $t$ , and  $p'_{i \rightarrow j}$  is its derivative. Each entry  $\sigma_{ij}$  in the infinitesimal generator is infact the *execution rate* of the process in that particular state.

An  $N$ -dimensional SAN consists of  $N$  stochastic automata that operate more or less independently of each other. The number of states in the  $k^{\text{th}}$  automaton is denoted by  $n_k$ ,  $k = 1, 2, \dots, N$ . The main objective of the following derivations, is the computation of the stationary probability distribution  $\pi$  of the overall  $N$ -dimensional system.

Given  $N$  independent stochastic automata, with associated infinitesimal generators  $Q^{(1)}, Q^{(2)}, \dots, Q^{(N)}$ , and probability distributions  $\pi^{(1)}(t), \pi^{(2)}(t), \dots, \pi^{(N)}(t)$  at time  $t$ , the *probability distribution* of the  $N$ -dimensional system,  $\pi(t)$ , is given by the tensor product<sup>1</sup> of the probability vectors of the individual automata at time  $t$ , that is:

$$\pi(t) = \bigotimes_{k=1}^N \pi^{(k)}(t) \quad (2)$$

To solve the  $N$ -dimensional system that is formed from independent stochastic automata, it suffices to solve the probability distributions of each individual stochastic automata and form the tensor product of these distributions. Although such systems may exist, for embedded system applications, we need to consider interactions among processes. There are two ways in which the stochastic automata can interact:

- A transition in one automaton forces a transition to occur in one or more other automata. These are called *synchronizing transitions* (events). Synchronizing events affect the global system by altering the state of possibly *many* automata. In any given automaton, transitions that are not synchronized are said to be *local transitions*.
- The *rate* at which a transition may occur in one automaton is a function of the state of another automata. These are *functional transitions*, as opposed to *constant-rate (non-functional)* transitions. Functional transitions affect the global system only by changing the state of a *single* automaton.

### The effect of synchronizing events

Let  $Q_i^{(k)}$  ( $k = 1, 2, \dots, N$ ) be the matrix consisting only of local transitions of any automaton  $k$ . Then, the part of the global infinitesimal generator that consists uniquely of local transitions can be obtained by forming the tensor sum of matrices  $Q_i^{(1)}, Q_i^{(2)}, \dots, Q_i^{(N)}$ . It has been shown in [21], that SANs can be always treated by separating out the local transitions, handling them in the usual fashion by means of a tensor sum and then incorporating the sum of two additional tensor prod-

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

ucts per synchronizing event. More than this, since tensor sums are defined in terms of the (usual) matrix sum (of  $N$  terms) of tensor products, the infinitesimal generator of a system consisting of  $N$  stochastic automata with  $E$  synchronizing events (and no functional transition rates) can be written as:

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

This quantity is referred to as the *global descriptor* of the SAN. It should be noted that, even if the descriptor can be written as a sum of tensor products, the solution is *not* simply the sum of the tensor products of the vector solutions of individual  $Q^{(i)}$ . This directly results from the fact that the automata are not independent.

### The effect of functional transition rates

Introducing functional transition rates has no effect on the *structure* of the global transition rate matrix other than, when functions evaluate to zero, a degenerate form of the original structure is obtained. So, although the effect of the dependent interactions among the individual automata prevents us from writing the solution as a tensor product of individual solutions, it is still possible to take advantage from the fact that the nonzero structure is unchanged. This is the motivation behind the *generalized tensor product* [8]. The descriptor is still written as in eqn. (3), but now the elements of  $Q_i^{(k)}$  may be functions. In this case, we replace each tensor product that incorporates matrices with functional entries with a sum of tensor products of matrices that incorporate only *average*

numerical entries. Then eqn. (3) becomes  $Q = \sum_{j=1}^T \bigotimes_{i=1}^N \bar{Q}_j^{(i)}$ ,

where  $\bar{Q}$  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. Although  $T$  may be large, it is bounded by  $T \leq (2E + N) \times \prod_{i \in F} n_i$ .

### An example

Consider the general *Producer-Consumer* paradigm with two interacting processes  $A$  and  $B$ . Process  $A$  (the *Producer*) writes data into an infinite buffer and process  $B$  (the *Consumer*) gets access to it (to read its contents), only if  $A$  is not writing any data. (We also note that the process  $A$  cannot write into the buffer when the process  $B$  is reading it.) Since the length of the buffer is infinite, no information is lost.

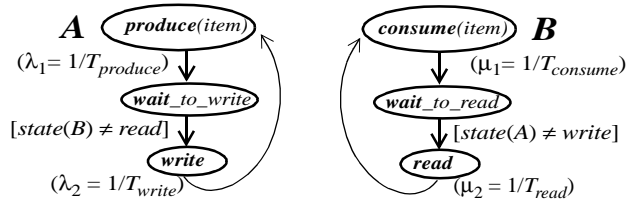


Fig.2: Interaction between two processes  $A$  (producer) and  $B$  (consumer)

Referring to process  $A$ , we observe a local transition between  $produce(item)$  and  $wait\_to\_write$  states; that is, this transition occurs at the fixed rate of  $1/T_{produce}$ , where  $T_{produce}$  is the time required to produce an item. The local part of the global generator ( $Q_l$ ) can be computed as:

$$Q_l = Q_l^A \otimes I_3 + I_3 \otimes Q_l^B \quad \text{where}$$

$$Q_l^A = \begin{bmatrix} -\lambda_1 & \lambda_1 & 0 \\ 0 & 0 & 0 \\ \lambda_2 & 0 & -\lambda_2 \end{bmatrix}, \quad Q_l^B = \begin{bmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & 0 & 0 \\ \mu_2 & 0 & -\mu_2 \end{bmatrix}, \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given an application,  $T_{produce}$  has a concrete meaning which can be deduced from a high-level analysis of the application. However, the transition from state  $wait\_to\_write$  to state  $write$  is functional, because it depends on the state of the other process. More precisely, this transition occurs if and only if the condition  $[state(B) \neq read]$  becomes true. Because of this dependency, we cannot associate a fixed rate to this transition; the actual rate depends on the overall behavior of the system. Finally, once the *Producer* gets access to the buffer, it transitions to its initial state (with the local transition rate  $1/T_{write}$ ). Similar considerations apply to the *Consumer* process.

We note that the complexity of the global generator increases very fast with the number of processes. However, as we shall see in the next section, it is possible to find out the probability distribution of the global system without building the global generator.

## 2.2 Performance model evaluation

Once we have the SAN model, we need to find out its *steady-state solution*. This means that the application exhibits some regularity and predictability in its behavior over the state space. This special condition is expressed by the equation

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

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

Our objective is then to solve eqn. (4) by using numerical methods that *do not* require the explicit construction of the matrix  $Q$  but can work with the descriptor in its compact form. When the global infinitesimal generator of a SAN is available in form (3), the numerical methods most suitable to solve (4) are iterative [9]. Thus, the underlying operation is the product of a vector with a matrix. Since

$$xQ = x \cdot \sum_{j=1}^T \bigotimes_{i=1}^N Q_j^{(i)} = \sum_{j=1}^T x \bigotimes_{i=1}^N Q_j^{(i)} \quad (5)$$

the main issue becomes the efficient computation of

$$x \bigotimes_{i=1}^N Q^{(i)}, \quad \text{where } x \text{ is a vector of length } \prod_{i=1}^N n_i.$$

For all practical purposes, it is sufficient to consider the case when  $Q^{(i)}$  contains only constant transition rates because, as explained in the previous subsection, this hypothesis covers the cases of

independent stochastic automata ( $T = N$ ), stochastic automata with synchronizing effects but no functional transitions ( $T = 2E + N$ ), and stochastic automata with synchronizing events and functional transitions, and the functional elements are handled by expansion ( $T > 2E + N$ ). Under this hypothesis,

the product  $x \bigotimes_{i=1}^N Q^{(i)}$  may be obtained using  $\prod_{i=1}^N n_i \times \sum_{i=1}^N n_i$  multiplications, where  $n_i$  is the number of states in the  $i^{th}$  automaton. (Typically, the  $n_i$  of the largest automata varies from a few to a few tens of states.) Due to the special structure resulting from the tensor product, computation is significantly improved using dynamic programming-like techniques.

Once the steady-state distribution is known, performance measures such as throughput, utilization, average response time can be easily derived. However, in order to calculate such performance figures, we need to find the *true rates* of the activities, which in turn requires that we calculate the probability that each activity is enabled. This is because, the specified rate of an activity is *not* necessarily the same as the rate of that activity in the equilibrium state since bottlenecks elsewhere in the system may slow the activity down. The true (or equilibrium) rate of an activity is thus specified by the rate multiplied by the *probability* that the activity is enabled, which means that the system is in that state in which it can perform that activity.

## 3. A case study: the MPEG-2 decoder

Our main observation is that, depending on the input traces, processes in the target application expose an inherent ‘clustering’ of probability distribution values in the steady-state regime. Moreover, depending upon the nature of the input traces, these processes react in different ways, giving scope for optimization and trade-offs. This is explained through the example of an MPEG-2 decoder (Fig.3). The decoder consists of the baseline unit, the Motion Compensator (MC), and the associated buffers. The baseline unit consists of the VLD (Variable Length Decoder), the IQ/IZZ (Inverse Quantization and Inverse Zigzag), the IDCT (Inverse Discrete Cosine Transform) and the buffer.

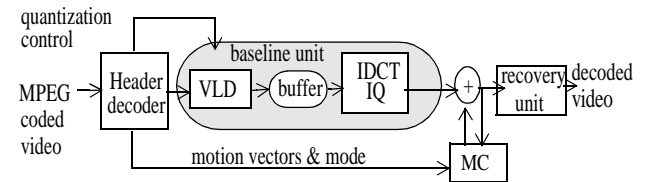


Fig.3 The block diagram of the MPEG-2 decoder

### 3.1 Modeling setup

We chose the Stateflow component of Matlab to model our system which uses the semantics of Statecharts, formally proposed by Harel [22]. To create the Stateflow model of the MPEG-2 video decoder, the *sequential* C-code of the decoder was split into several processes and the communication

among processes made explicit by using synchronization and functional transitions as in Section 2. To this end, we first identified the main modules of the application. These modules were then mapped to the corresponding code segments and functions. Then, from the application profiles, we identified the interaction between the modules and also developed the function sequence and call-graphs, to obtain a better view of the data and control flow inside the modules.

We model the process graph obtained from the application corresponding to the baseline unit following the *Producer-Consumer* paradigm. We described the VLD process as the *Producer* and the IDCT/IQ unit as the *Consumer*. The VLD decodes the input stream, generates macroblocks and puts them into the buffer. These are picked up by the *Consumer* to compute IDCT's and output data to reconstruct the frames. Both the buffer access and the IDCT output need to access the memory which is a shared resource. To unravel the full potential for efficient mapping offered by applications, we assume that each process has its own space to run, and does not compete for any computing resources. The buffer is also mapped to a queue, to allow us to see its utilization characteristics.

The model of the application was evaluated using the analytical procedure in Section 2 under two scenarios, using the *sf2san* tool that we developed for exploiting the SAN analysis techniques. The tool works within Matlab environment and constructs the infinitesimal generator matrices corresponding to each automata from the Stateflow diagrams. This involves deriving the matrices corresponding to synchronization and functional transitions, apart from those corresponding to local transitions. These transition matrices incorporate rate informations, provided by the designer from trace-driven simulations of the application. Using the formalism provided in Section 2 the tool is able to obtain the steady-state probability distributions of the automata for all the states.

### 3.2 Results and discussion

In our experiments, we assume that the bit-rate of the input video stream is the same for all scenarios. Three different runs are presented: in the first one, the *Producer* produces at a rate equal to the rate at which the *Consumer* consumes, on average. The second run, corresponds to a restructured version of the decoder, where the *Producer* is slightly slower than it is in the first run. The third one corresponds to another concurrent version of the decoder (differently structured) where the *Producer* is slightly faster than it is in the first run. Moreover, two scenarios are presented for comparison:

- The **first scenario** corresponds to the case, of running a clip of with very low correlation (<10%) between adjacent macroblocks and frames (e.g., playing an MPEG-coded video movie like *Terminator2*). This low correlation appears because the adjacent macroblocks in the I-frames can be very different. Further, due to a lot of special effects, the error between the predicted frame and the original frame (coded using P and B frames) can be very high.

The figures corresponding to the steady-states probabilities of some of the states are given in Fig.4, for three runs. The first column in the *Producer* state diagram shows the probability of the VLD process using the CPU ( $CPU_V$ ). The second column shows the probability that the process is waiting for the buffer ( $Wait_B$ ) because the *Consumer* process is accessing it. The third column shows the probability of the *Producer* process being blocked because the buffer is full ( $Full_B$ ). For the *Consumer*, the first column refers to the probability of the process being blocked by an empty buffer ( $Empty_B$ ). The second column shows its probability of using the CPU ( $IDCT_W$ ) that would be obtained assuming no correlation between macroblocks. The third shows the actual probability of the *Consumer* process using the CPU that is depending on the input patterns, ( $CPU_A$ ). The columns in the *Buffer* state show the probability distribution of the length of the buffer ranging from 0 to 4.

In run one of the first scenario, we observe that the *Producer* is active only 70% of the time, and it is waiting for the buffer 30% of the time, while the *Consumer* is waiting because the buffer is empty for 32% of the time. In the second run, we find that the *Producer* is active 88% of the time, because the *Consumer* is consuming at a faster rate. Moreover, the *Consumer* remains in the wait state for 60% of the time because the buffer is empty. We observe that slowing down the *Producer* improves CPU utilization (as CPU-time = 0.88 in second run). In the third run, where the *Producer* is producing at a slightly faster rate, we observe that not only is it blocked because the *Consumer* is accessing the buffer, but also because the buffer is full. Moreover, it remains blocked for more than 50% of the time.

- The **second scenario** corresponds to running a clip of with very high correlation (>90%) between adjacent macro-blocks and frames (e.g., playing an MPEG-coded video of the stock-exchange report or weather channel). In terms of steady-state probabilities, because of the very different nature of the input stream, we see a very different set of plots in Fig.5 compared to the previous case (Fig.4).

Comparing Fig.4 and Fig.5, we observe that there is not much change for the *Producer* (most probably because the bit-rate is same). However, the CPU-active time for *Consumer* changes drastically depending on the inputs. As the correlation increases, fewer IDCT's need to be computed because of very little change in the scenes of the video stream. Further, it is interesting to note that the buffer requirements actually increase in the second scenario. Although the computation on average is low, abrupt changes in the scenes of the movie cause an increase of the buffer utilization. These are reflected in the buffer length changes, particularly noticeable when we reduce the speed of computation of *Consumer* in the third run, after seeing the low CPU-utilization. Further, we observe that the buffer length is about 1.33 on an average, and may eventually reach length 4 (as would be predicted by a worst-case analysis) in less than 5% of the time.

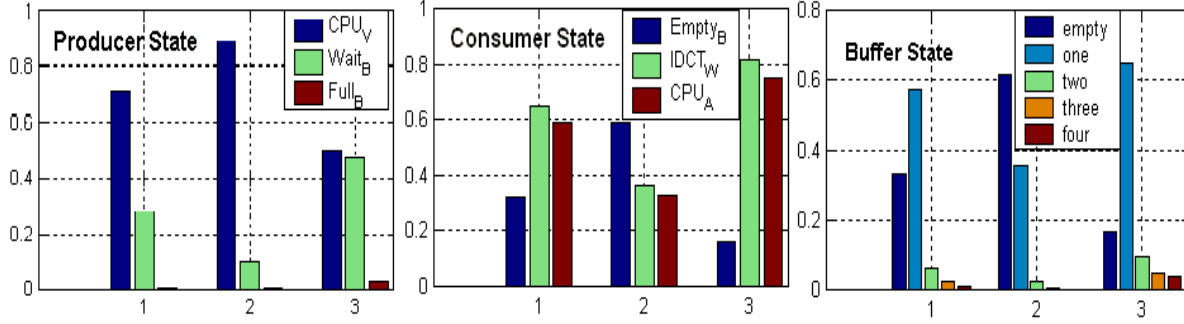


Fig.4. Probability distribution for *Producer*, *Consumer* and *Buffer* in Scenario 1

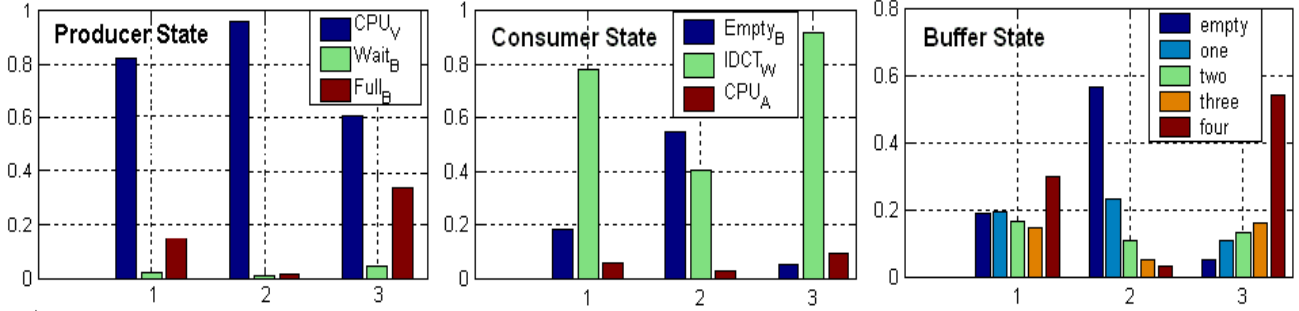


Fig.5. Probability distribution for *Producer*, *Consumer* and *Buffer* in Scenario 2

#### 4. Implications in the design process

Once we know the steady-state regime, by using the proper cost functions for each state, the performance measures of interest can be easily derived. Since there are no limitations on the cost functions that can be attributed to the states, this kind of analysis has a lot of potential. For instance, the utilization of the buffer (channel) can be easily computed from the average number of frames weighted with their corresponding probabilities. It must be emphasized that, to calculate performance figures (like throughput, latency), we need the *true* rates for the activities. For instance, in the second scenario, although the data from the IDCT state of the *Consumer* gets directly displayed at the specified display rate  $\lambda_C$ , the actual (*true*) display rate will be only  $0.07 \cdot \lambda_C$ ; that is, 14 times less than the specified rate  $\lambda_C$  when the IDCT is assumed to run in a stand-alone mode. This is the correct rate that the designer should use during the optimization process.

##### 4.1 System-level power estimation

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:

$$P^{(k)} = \sum_{all\ i} \pi_i \cdot P_i + \sum_{all\ i,j} \lambda_{ij} \cdot P_{ij} \quad (6)$$

where  $P_i$  and  $P_{ij}$  represent the power consumption per state and per transition, respectively, and  $\pi_i$  is the steady-state probability and  $\lambda_{ij}$  is the transition rate associated with the transitions between states  $i$  and  $j$ . Having already determined the solution of eqn. (4), the  $\pi_i$  value (for a particular  $i$ ) can be

found by summing up the appropriate components of the global probability vector  $\pi$ . The  $P_i$  and  $P_{ij}$  costs are determined during an off-line pre-characterization step where other proposed techniques can be successfully applied [13].

To obtain the power values, we simulated the MPEG-2 decoder, using the Wattch [23] architectural simulator that estimates the CPU power consumption based on a suite of parametrized power modes. By specifying a low-power *Strong-Arm* like processor, we obtained an average power value of 4.6W for the VLD module, and 4.8W for the IDCT. Using these the power figures, it is easy to obtain the average power characterization for the entire system under varying loads. This is useful to trade-off performance and power. In our example, using eqn. (6), we have obtained the average power values of 3.5W (*Producer*), 3.2W (*Consumer*), for scenarios 1 and 2, respectively. Furthermore, we can multiply these power values with the average buffer lengths from Figs. 4 and 5 (0.66 and 2.0, respectively), and get the *powerxdelay* characterization of the system; that is, 4.42J for scenario 1 and 8.26J (almost double!) for scenario 2.

##### 4.2 Mapping Applications based on Clustering effects

The second aspect that we would like to consider is the effect of probabilities clustering, as a function of the characteristics of the input video stream. To study this, we consider the global probabilities rather than the local probabilities that we have presented so far. The global probabilities present the probability of the whole system being in a certain configuration. For example, a configuration may consist of

the *Producer* being blocked because the *Buffer* is full or the *Consumer* is reading the *Buffer*.

For simplicity, we map the states of the processes into two broadly defined states R (run) and I (idle). The process is in state R if it is actively computing and using the CPU, otherwise it is in state I. This means that for the *Producer* process the state ( $CPU_V$ ) gets mapped into R, while all other states mentioned get mapped into I.

In Table 1, we present some of the important global probabilities corresponding to two runs (A and B), and show their importance in the mapping process. Runs A and B are two typical cases where the *Producer* and the *Consumer* have balanced rates and the input stream corresponds to scenario 1 and 2 respectively. (The local probabilities for these cases have been presented as Run 1 for scenario 1 in Fig.4 and Run 1 for scenario 2 in Fig.5.)

In Table 1, the first row shows that the both the *Producer* and *Consumer* are running simultaneously with probability 0.26 for run A, and 0.60 for run B. The second row shows that the *Producer* is running alone with probability 0.62 for run A and 0.22 for B, while the third row shows that both the processes are idle with probability 0.11 for run A and 0.12 for run B.

Probabilities		Producer	Consumer	Buffer
Run A	Run B			
0.26	0.60	R <sup>a</sup>	R	0-1
0.62	0.22	R	I	0-1
0.11	0.12	I <sup>b</sup>	I	0-1
0.01	0.01	All others		

Table 1: Global Probability Distribution

- a. R means the process is actively computing
- b. I means the process is idle and is not computing

Having this information about the global probabilities, we can put processes with similar running probabilities in the same cluster, and define a mapping of such processes to a set of architectural resources. For instance, in our example, we may want to assign all processes that are active more than 50% of the time (that is, probability greater than 0.50) onto a low-power processor, and those which are active around 25% of the time into some ASIC. Using such a strategy, the processes can be mapped onto the processors as shown in the Fig.6. (The size of the states of the processes in the figure is proportional to the value of probability of being in that state.)

The above case is especially useful if the designer is ready to have a processing element for each process. However, the number of resources/processing elements available may be less than the number of processes. It turns out that the derived steady-state probabilities can help us to partition the set of automata based on the available resources, such that processes mapped onto a particular processing element compete minimally. From Table 1, we observe that in Run A, the probability that the *Consumer* is in state R and the *Producer* in state I, simultaneously, is greater than 0.6, while the probability of both being in state R is 0.26. This may be useful to

the designer, to map processes that compete with each other onto different processing elements as shown in Fig.7, and assign the corresponding task priorities so that they can be scheduled without too much overheads.

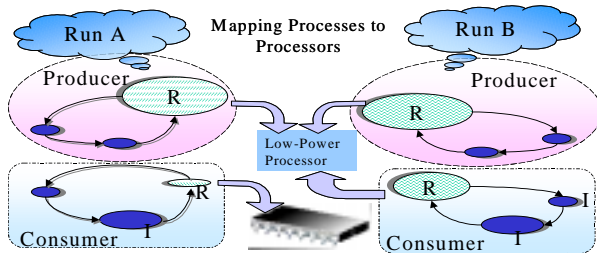


Fig 6. Mapping processes to processing elements using first strategy

Finally, to complete the discussion, we present in Fig.8 the global picture of resource utilization for 36 runs in the two scenarios discussed above and a third scenario where the correlations were set in-between. For each run we observe the global state probabilities, as a permutation of VLD state (*Running\_CPU*, *Idle\_CPU*), with IDCT state (*Running\_CPU*, *Idle\_CPU*) and *Buffer* state ( $\{0,1\}, \{2,3\} \{4\}$ ).

From Fig.8, we see that, from a large number of cases, S4 is the predominant one. S4 corresponds to the case (R, I, 0-1), i.e., VLD is in *Running\_CPU* state, IDCT is in *Idle\_CPU*, and the *Buffer* is either empty or contains one item. Moreover, it shows that S5 to S9 and S11 are very rare. This indicates that the two components can be mapped onto the same processing elements.

On the other hand, S12 shows that if the buffer speed is greatly reduced (as was done for some runs), the (I, I, 4) state becomes prominent. From the figure it is possible to get an idea as to what bus/buffer speed should we choose such that the power/performance trade-off becomes acceptable. Similarly, S10, corresponding to (I, I, 0-1) is seen to predominate in some cases, indicating that the system resources exceed the requirement, and then the system is idle.

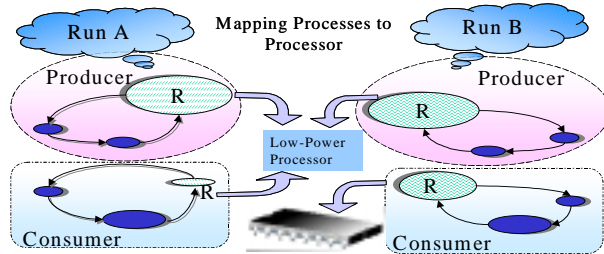


Fig 7. Mapping processes to processing elements using second strategy

It is easy to see that this approach can be very useful, in more complicated cases, involving a larger number of automata. This is of interest to any system designer wanting to trade-off power and computing speed. We point out that, unlike the local probabilities, the *global probabilities* give a much better insight into the behavior of the system and should be used for optimization purposes.

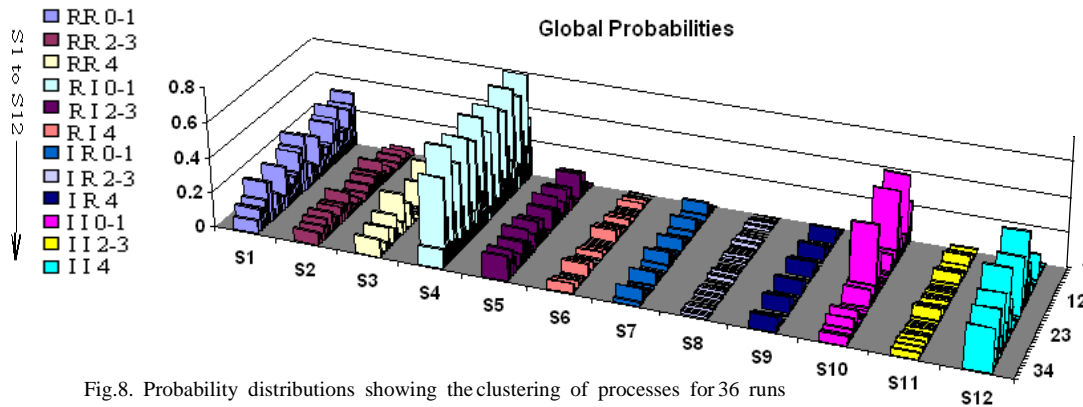


Fig.8. Probability distributions showing the clustering of processes for 36 runs

Finally, the CPU time needed for our analysis is less than a few seconds in each of the runs, while the error remains smaller than 5% compared to the simulation results. This is at least 2-3 orders of magnitude faster than the active simulation time required to obtain the same results. Hence, the approach can significantly cut down the design cycle and, at the same time, enhance the opportunities for exploring the entire design space.

## 5. Conclusion

We have presented a formal technique for system-level analysis based on SANs. Using this formalism, we have shown that, under various input traces, the steady-state behavior of the application can be characterized by very different ‘clustering’ of the probability distribution values. This information is essential for obtaining efficient mappings of the applications onto a chosen platform. Experimental results have been presented for an MPEG-2 video decoder.

## 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] D.D. Gajski, F. Vahid, S. Narayan, J. Gong, ‘Specification and Design of Embedded Systems,’ Prentice Hall, 1994.
- [3] R.K. Gupta, ‘Co-synthesis of Hardware and Software for Digital Embedded Systems,’ Kluwer Academic Publishers, 1995.
- [4] W. Wolf, ‘Hardware-software co-design of embedded systems’ in *Proc. IEEE*, Vol. 82, July 1994.
- [5] A. Sangiovanni-Vincentelli, B. Kienhuis, J. Rabaey, K. Vissers, and D. Verkest, ‘System Level Design with Embedded Platforms,’ *Tutorial DAC*, Los Angeles, CA, June 2000.
- [6] A. Kalavade, P. Moghe, ‘A tool for performance estimation of networked Embedded End-Systems,’ *Proc. DAC*, San Francisco, CA, June 1998.
- [7] P. Lieverse, P. Van der Wolf, E. Deprettere, and K. Vissers, ‘A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems,’ to appear in *Journal of VLSI Signal Processing*.
- [8] B. Plateau, J.M. Fourneau, ‘A Methodology for Solving Markov Models of Parallel Systems,’ *Journal of Parallel and Distributed Comp.*, Vol. 12, 1991.

- [9] W.J. Stewart, ‘An introduction to the Numerical Solution of Markov Chains,’ Princeton University Press, New Jersey, 1994.
- [10] S. Malik, M. Martonosi, Y.-T. Li, ‘Static Timing Analysis of Embedded Software,’ in *Proc. DAC*, Anaheim, CA, 1997.
- [11] W. Ye, R. Ernst, T. Benner, J. Henkel, ‘Fast Timing Analysis for Hardware-Software Cosynthesis,’ *Proc. Intl. Conf. Computer Design*, 1993.
- [12] T.-Y. Yen, W. Wolf, ‘Performance Estimation for Real-time Distributed Embedded Systems,’ *Proc. Intl. Conf. Computer Design*, 1995.
- [13] T. Simunic, L. Benini, G. DeMicheli, ‘Cycle-Accurate Simulation of Energy Consumption in Embedded Systems,’ *Proc. DAC*, New Orleans, June 1999.
- [14] T. Zhou, X. Hu, and E. Sha, ‘A Probabilistic Performance Metric for Real-Time System Design,’ *Proc. CODES*, May 1999.
- [15] H. Hulgaard, S.M. Burns, T. Amon, G. Borriello, ‘An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems,’ *IEEE Trans. on Comp.*, Vol. 44, no. 11, 1995.
- [16] 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.
- [17] J. Buck, S. Ha, E.A. Lee, D.G. Messerschmitt, ‘Ptolemy: A framework for simulating and prototyping heterogeneous systems,’ *Intl. Journal of Comp. Simulation*, vol. 4, Apr. 1994.
- [18] F. Balarin et al, ‘Hardware-Software Co-design of Embedded Systems - The POLIS approach’, Kluwer Academic Publishers, 1997.
- [19] P. Chou, R.B. Ortega, and G. Boriello, ‘The Chinook Hardware/Software Co-Synthesis System,’ *Proc. Intl. Symposium on System Synthesis*, 1995.
- [20] J. Buck, R. Vaidyanath, ‘Heterogeneous Modeling and Simulation of Embedded Systems in El Greco,’ *Proc. CODES*, March 2000.
- [21] B. Plateau, ‘On the stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms,’ *Proc. ACM Sigmetrics*, Aug. 1985.
- [22] D. Harel, ‘Statecharts: A visual formalism for complex systems,’ in *Sci. Comp. Prog.*, Vol. 8, 1987.
- [23] D. Brooks, V. Tiwari and M. Martonosi, ‘Watch: a framework for architectural-level power analysis and optimizations,’ *Proc. ISCA*, June 2000.