Formal Analysis and Validation of Continuous-Time Markov Chain based System Level Power Management Strategies

Gethin Norman [*] David Parker[*] Marta Kwiatkowska[*] Sandeep K. Shukla [**]
Rajesh K. Gupta [†]

[*] School of Computer Science, University of Birmingham,
Birmingham B15 2TT, UK

[**]Bradley School of Electrical and Computer Engineering,Virginia Tech
Blacksburg, VA 24060, USA

[†] Department of Computer Science and Engineering, University of California at San Diego
La Jolla, CA, USA

Abstract

We have shown in the past that competitive analysis based power management strategies can be automatically analyzed for proving competitive bounds and for validating power management strategies using the SMV model checker. In this paper, we show that stochastic modelling based strategies for power management can similarly be automated for computing optimal strategies. Further, these can be analyzed for finding system parameters for satisfying probabilistic constraints. Effects of any changes in probabilistic assumptions can be easily analyzed without expensive and time consuming simulations. We demonstrate our methodology using the probabilistic model checker PRISM. We model the system using a continuous-time Markov chain, and compute strategies under varying requirements for performance. We also prove probabilistic properties of strategies using PRISM, which gives insight into individual strategies and pragmatics of their implementations. We also show the effects of changing probabilistic assumptions computed by our method and compare the results with other stochastic analysis based methods, and show that we obtain similar results in a uniform framework of probabilistic model checking.

1 Introduction

Power Management is an important area of research [1, 3, 4, 16, 19, 21] because of an increasing trend in the usage of portable, mobile, and hand-held electronic devices. These devices usually run on batteries and any savings in power usage translate to extended battery life. Various approaches to low power design have been explored, at device, circuit and micro-architecture level. System level power management exploits application characteristics and manages various system devices for power optimization. System components such as network interface cards, disk drives and DRAM, are manufactured with a number of power modes which can be changed by an operating system through standard APIs such as ACPI [9] and power-aware API [14]. However, in order to take advantage of these power modes and APIs, the power management strategies need to be implemented at the O/S level. In this context, Dynamic Power Management (DPM) strategies refer to strategies that attempt to make power mode changing decisions based on the information about their usage pattern available at runtime. The objective of such strategies is to minimize power consumption, while minimizing the effect on performance.

In recent years, several stochastic modelling based approaches for designing DPM strategies have been proposed [2, 1, 6, 16, 17]. Irrespective of whether these models are based on stationary discrete-time Markov chains [13], continuous-time Markov chains [16] or their variants [23], the methodologies depend on modelling the input arrival process and the behaviour of the power managed components. This is done by manually creating the stochastic matrices or generator matrices for these processes, and then formulating optimization problems whose solution is the required strategy. In [18], the power manager and managed components are modelled using stochastic Petri nets. This allows automatic generation of the stochastic matrices and formulation of the optimization problems. These *exact* optimization problems are meant to optimize the *average* energy usage while minimizing the *average* number of requests waiting to be served. They are usually validated by simulation to check the soundness of the modelling assumptions and the effectiveness of the strategies in practice [16, 13].

In this paper we present an innovative approach that exploits probabilistic model checking to formally validate the effectiveness of a stochastic DPM strategy that obviates the need for exten-

sive simulations. We demonstrate the utility of our approach using one example system taken from [17] based on continuous-time Markov chains. Probabilistic model checking is a relatively new technology [7, 5] which can formally validate various probabilistic properties of stochastic models. In the power management context, the request arrival process, uncertainties in service times, and uncertainties in time taken between issuing a power management command and its effect can be modelled very naturally. Probabilistic model checking allows formal exhaustive analysis of the design space for the strategies and proves *probabilistic guarantees* on the behaviour of the stochastic management strategies.

There are several advantages of this approach. It helps in designing strategies in the model checker framework, thereby employing an existing technology to the development and analysis of stochastic power management strategies. Moreover, since probabilistic model checking is inherently exhaustive in its search among all possible scenarios, more useful information can be obtained about the design space than using simulation. For example, optimal buffer sizes, average delays, probabilities of various corner case scenarios etc, and probability based comparisons between various delay-cost possibilities (obtainable by competing DPM strategies) can easily be predicted. In contrast, existing approaches cannot provide such information in any direct way because they only optimize the average case.

In particular, in this paper, we show how to obtain the optimal DPM policy under different performance constraints and then compute both the average power consumption, and the average number of requests awaiting service, when probabilistic assumptions are varied and the stochastic policy derived under ideal assumptions are held fixed. This allows us to avoid extensive simulation as done in [17], and also allows us to derive other figures of merit and extremities.

Due to lack of space, this extended abstract briefly illustrates our modelling methodology and our approach to derivation and validation of the derived strategies.

2 System Level Power Management

Power management in embedded computing systems is achieved by actively changing the power consumption profile of the system by putting its components into power/energy states which are sufficient to meet functionality requirements. For example, an idling component (such as a disk drive) can be put into a slow-down or shutdown state. Of course, bringing such a component into an active state may then require additional energy and/or latency to service tasks. The problem we consider here is as follows: *given a power managed component, such as a disk drive or a network interface card, derive a randomized power management strategy which minimizes average power dissipation, under the constraint that the average number of requests awaiting service is bounded by an a priori constant.* We also seek to develop probabilistic guarantees on the worst case and best case scenarios, enabling us to quantify the effectiveness of the DPM strategy. These measures include the worst case delay and worst and best power consumption. We also want to obtain information on buffer sizes and other design space parameters (in a probabilistic sense). In other words, we want to know the values of these parameters as a function of probabilities of certain delays or power consumption in the system.

2.1 Related Work on Dynamic Power Management (DPM)

Dynamic Power Management (DPM) attempts to make optimal decisions (usually under the control of the operating system) at runtime based on dynamically changing system state, functionality and delay requirements [8, 24, 4, 3, 22, 20, 6, 21]. A survey of DPM techniques can be found in [1] which classifies DPM strategies into two main groups: (a) *predictive schemes*, and (b) *stochastic optimum control* schemes. Predictive schemes attempt to predict the timing of future input to the system and based on such predictions, schedule shutdown (usually to a single lower power state) of the system. Stochastic optimum control is a well-researched area [1, 23, 2, 6, 16, 17]. The chief characteristic of these approaches is the construction (and validation) of a mathematical model of the system that lends itself to a formulation of a stochastic optimization problem, and then creation of strategies to guide the system power profile that achieves the highest power savings in the presence of the uncertainty related to system inputs.

While several useful and practical techniques have been developed using predictive and stochastic optimum control schemes, as of now it is difficult to develop bounds on the quality of these results without extensive simulations and/or model justification.

2.1.1 Stochastic Control and DPM

Stochastic control oriented dynamic power management work [2, 16] has relied on modelling inter-arrival times using an exponential distribution. In practice, such stochastic modelling seems to work well for specific kinds of applications. However, the approaches varied in the modelling of time. For example, in [2] the arrival process and service process are all modelled as discrete time Markov chains, whereas in [16], they are modelled with continuous time Markov chains. In more recent work, such as in [23], extended models which incorporate more general stochastic processes for modelling event arrivals have been considered. In this paper, we focus on continuous-time Markov chains following [17].

2.2 Power States of Components

Due to the importance of the minimization of power consumption in today's embedded systems, a lot of work has been initiated in both component manufacturing industry, and the systems design industry. Notably, Intel, Toshiba, and Microsoft's ACPI [9] standardizes APIs to access and control devices' power states. A similar industrial effort is seen in OnNow [12]. Device and component manufacturers also provide multiple power saving states which can be controlled under the operating system through these standardized APIs. Table 1-3 shows the data from [17] for a 3-state (busy, idle and sleep) device. It illustrates the power states of a portable hard disk drive and service time and power dissipation averages, which are used in our experiments.

state	“sleep”	“idle”	“busy”
Avg. Power(Watt)	0.13	0.95	2.15
Avg. Service Time(sec)	0	0	0.008

Table 1. Average Power Consumption Values and Service Times for the managed device [17].

Avg. Energy(Joules)	“sleep”	“idle”	“busy”
“sleep”	0	7.0	–
“idle”	0.067	0	0
“busy”	–	0	0

Table 2. Average Energy Consumption Values for state transition of the device [17].

3 Probabilistic Model Checking

At a high level of abstraction, a model can often be simplified for instance by replacing determinism by nondeterminism. However, complete nondeterminism often leads to an inability to prove any useful property of such systems. As a result, probabilities are often used to abstract some of the low level disregarded information, and a quantified nondeterminism can be represented in a probabilistic model. For example, consider the service time by a disk-drive per request; if one models the functionality of the disk-drive, the queues, the environment, the device drivers and the operating system and the architecture, one may be able to predict exactly for each request how much service time is required. However, modelling disk-drive behaviour in detail is difficult and not amenable to formal analysis. However, by observing its behaviour for a sufficiently long time, one can infer that the disk-drive exhibits probabilistic behaviour with certain parameters. As an example, the service time per request is often modelled as an exponential distribution with a mean determined by sampling. Such information is often useful in modelling and analysing this behaviour and also in devising probabilistic algorithms for managing such devices.

3.1 Probabilistic Model Checking and PRISM

Probabilistic model checking refers to a state space analysis technique for probabilistic finite state systems. The system is usually specified as state transition system, with probability measures on the rate of transitions, and a probabilistic model checker applies algorithmic techniques to analyse the state space and calculate the probabilities of reaching different states etc. Given probabilistic assertions about the system, such model checkers can prove or disprove such assertions by means of algorithmic techniques [7, 5].

Avg. Transition Time(sec)	“sleep”	“idle”	“busy”
“sleep”	0	1.6	–
“idle”	0.67	0	0
“busy”	–	0	0

Table 3. Average Transition Times for state transition of the managed device [17].

We use PRISM [11, 15], a probabilistic model checker developed at the University of Birmingham. It supports analysis of three types of probabilistic models: discrete-time Markov chains, continuous-time Markov chains and Markov decision processes. These models are described in a high-level language based on guarded commands with probabilistic information attached to them. Properties of the models to be analysed are specified in the probabilistic temporal logics PCTL and CSL. This allows us to express various probabilistic properties such as “some event happens with probability 1”, and “the probability of cost exceeding C is 95%”. The model checker then analyses the model and checks if the property holds in each state.

We model the managed component’s servicing behaviour, the request arrival process, the buffers where requests are queued, and the strategies that make the servicing component move between power states. Then, through analysis, obtain quantified information regarding the effectiveness of alternative strategies, and other useful information as will be seen in the rest of the paper. Note that in this approach we do not rely on the simulation results to find the effectiveness of various approaches. Irrespective of whether they are based on stationary discrete-time Markov chains, continuous-time Markov chains or their variants, existing methodologies depend on modelling the input arrival process and the behaviour of power managed components by creating the stochastic matrices or generator matrices for these processes by hand, and then creating and solving optimization problems from those to optimize the average case. One novelty of this work is that we express the behaviour of the input generator and power managed component, as well as the power manager, in a high level probabilistic language for expressing stochastic state machines. This allows generation of the matrices, and carry out the rest of the required computation for designing strategies in a model checker framework.

4 Implementation and Results

4.1 The System Model

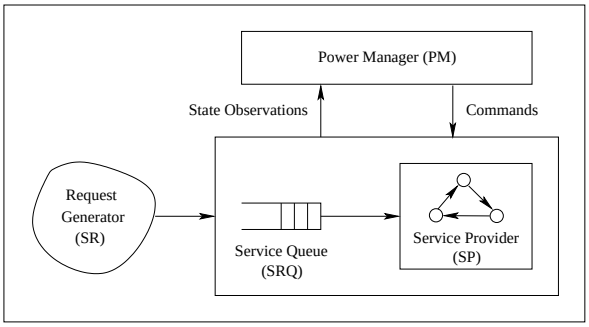

Figure 1. The System Model.

We consider the system model as illustrated in Figure 1. The model consists of a Service Requester (SR), a Service Provider

(SP), Service Request Queue (SRQ), and the power manager (PM). In both discrete-time as well as continuous-time models, this level of description is the same. However, they differ in how time is represented. In [17], time is considered to be continuous and mode switching commands can be issued at any time, and events can happen at any time.

4.2 Model Construction

We have designed generic models of the the Power Management system in PRISM's input description language, for the continuous time model of [17]. Then, using PRISM, we are able to construct the full generator matrix of the system, and hence construct the optimization problem whose solution is the optimal policy. Moreover, once the optimal policy is found, by using the generic description we can construct a model of the system corresponding to this policy and investigate its performance. Table 4 shows our derived optimal policies for the 3 state device from [17]. Varying the average number of requests awaiting service (queue size) constraints, we obtain different stochastic policies. The left-most column lists the average queue size constraints allowed, and right column summarizes the power manager's policy. This calculation is done by generating the matrices in PRISM and formulating and solving the linear optimization problem in MAPLE symbolic solver.

Figure 2, 3 and 4 show the PRISM representation of a 3 state continuous-time Markov chain model, after deriving a policy under the performance constraint of 1 (average number of requests awaiting service). This model shows a power manager with probabilities taken from the fourth row of Table 4. This model can be modified by just modifying the PM module, using a different row from Table 4. This way we analyse performance under varied constraints.

```
// POWER MANAGER performance constraint: average size of queue ≤1
// if the queue is full and SP is off then go to idle
// if the queue is empty and the SP is on then
// go to sleep with probability 0.008963 and stay in idle with probability 0.991037

module PM
    pm : [0..1];
    // 0 – loop or go from sleep to idle
    // 1 – try and go from idle to sleep (only do this if queue is empty)

    // when queue is full go from sleep to idle
    [sleep2idle] q = QMAX → pm′ = pm;
    // probabilistic choice when queue becomes empty
    [serve] q = 1 → 0.008963 : pm′ = 1; // go to sleep
    [serve] q = 1 → 0.991037 : pm′ = 0; // stay in idle
    [serve] !q = 1 → pm′ = pm; // loops for the remaining cases
    [idle2sleep] pm = 1 → pm′ = 0; // idle to sleep
    [request] true → pm′ = 0; // reset p when queue no longer empty

endmodule
```

Figure 2. Encoding of the Power Manager in PRISM for the derived Policy under the performance constraint = 1.

```
// SERVICE PROVIDER simple model with 3 states (sleep, idle and busy)
// ASSUMPTIONS
// (i) SP automatically moves from idle to busy when ever a request arrives
// (ii) moves from busy to idle when ever a request is served
// (iii) transitions between the sleep and idle states are controlled by the PM

// rates of local state changes
rate idle = 10/16; // sleep to idle
rate sleep = 100/67; // idle to sleep
rate service = 1000/8; // rate of service

module SP
    sp : [0..2]
    // 0 – sleep, 1 – idle and 2 – busy

    // SLEEP TO IDLE TRANSITIONS
    // something in the queue so start serving immediately
    [sleep2idle] sp = 0 ∧ q > 0 → idle : sp′ = 2;
    // nothing in the queue so start go to idle
    [sleep2idle] sp = 0 ∧ q = 0 → idle : sp′ = 1;
    // IDLE TO SLEEP
    [idle2sleep] sp = 1 → sleep : sp′ = 0;
    // IDLE TO BUSY (when a request arrives)
    [request] sp = 1 → sp′ = 2;
    [request] !sp = 1 → sp′ = sp;  // loop for other states
    // SERVE REQUESTS
    [serve] sp = 2 ∧ q > 1 → service : sp′ = 2; // queue not empty: stay in busy
    [serve] sp = 2 ∧ q = 1 → service : sp′ = 1; // queue empty: go to idle

endmodule
```

Figure 3. Service Provider Module representation in PRISM.

```
// SERVICE REQUESTER AND QUEUE (single priority queue)
const QMAX = 20; // maximum size of queue
rate arrive = 100/72; // arrival rate (mean value = 0.72)

module SQ
    q : [0..QMAX];  // number of requests awaiting service

    // request arrives
    [request] true → arrive : q′ = min(q+1, QMAX);
    // request is served
    [serve] q > 0 → q′ = q − 1; //

endmodule
```

Figure 4. Service Queue Module representation in PRISM.

4.3 Model Anaylsis

PRISM allows us to compute performance measures for both the CTMC and DTMC models, including: the long run average cost; the long run average number of requests awaiting service; and the probability that the queue does not exceed a certain size within a given time bound.

We are currently developing a prototype tool extending the modelling capabilities of PRISM to include expected time and expected cost. This will allow us, for example, to calculate the expected cost/queue size within a given time bound. Furthermore, we are also considering the use of more general distributions, such as Pareto or uniform, which can be used to give a more realistic model of the inter-arrival time of service requests.

Table 5 illustrates the performance of this power manager under varying performance constraints, when the arrival process also

constraint	policy
average queue size ≤ 20	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep
average queue size ≤ 10	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep with probability 0.869135 and stay in idle with probability 0.130865
average queue size ≤ 5	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep with probability 0.075140 and stay in idle with probability 0.924860
average queue size ≤ 1	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep with probability 0.008963 and stay in idle with probability 0.991037
average queue size ≤ 0.25	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep with probability 0.002014 and stay in idle with probability 0.997986
average queue size ≤ 0.1	if queue is full and SP is off, then go to idle if queue is empty and the SP is in idle, then go to sleep with probability 7.39e-04 and stay in idle with probability 0.999261

Table 4. Policies under varying constraints on the average queue size derived using PRISM.

varies from the ideal exponential ones. These numbers are computed using steady state probabilities and, in the non-exponential cases, the techniques presented in [10].

As can be seen in Table 5 the average queue size when requests arrive with a Pareto distribution are in general much smaller than when requests arrive with the other distributions considered. This is due to the Pareto distribution's *heavy tail*, which means that, in the long run, many requests will not arrive for a very long time, and hence in these cases the service provider (SP) will serve all pending requests, and then the system will spend a long time with the queue empty. While for the remaining distributions, Table 5 shows that the long-run performance and costs are reasonably close to that of an exponential arrival process.

5 Conclusion and Future Work

We show that probabilistic model checking can be effectively used to obtain stochastic power management policies, by taking an example of a 3-state managed device and modelling the arrival process and service distribution as exponential with parameter taken from known experiments [18]. We also obtain the power management policies under varied performance constraints (rather than only the one that minimizes average number of requests awaiting service). We also evaluate performance of policies in terms of average queue size and energy expenditure, if the arrival processes are not ideal (exponential) but uniform, Pareto, Erlang, and deterministic. This is useful in the design and effective analysis of system level DPM strategies without having to do exhaustive simulation. Simulation is not only time consuming, but also does not guarantee the quality of results quantitatively. Ongoing work focuses on building an analytical framework that includes both continuous time and discrete time Markov chain analysis, as well as more generic process characteristics.

6 Acknowledgements

This work was supported by NSF grant CCR-0098335, EPSRC grant GR/N22960, QinetiQ, SRC, and DARPA/ITO supported PADS project under the PAC/C program. Part of the work was possible by a visiting fellowship from University of Birmingham.

References

[1] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, 8(3):299–316, 2000.

[2] L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.

[3] L. Benini, G. De Micheli, and E. Macii. Designing Low-power Circuits: Practical Recipes. *IEEE Circuits and Systems Magazine*, 1(1):6–25, Mar. 2001.

[4] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Publications, 1998.

[5] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[6] E. Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Proceedings of the Design Automation and Test Europe*, 1999.

[7] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

[8] C.-H. Hwang, C. Allen, and H. Wu. A Predictive System Shutdown Method For Energy Saving of Event-Driven Computation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 28–32, 1996.

[9] Intel and Microsoft and Toshiba. Advanced Configuration and Power Interface Specification. Website, December 1996.

[10] M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. In *2nd Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability*, 2002. To appear.

performance constraint	performance measure	inter-arrival time distribution				
		exponential	deterministic	Erlang10	uniform	Pareto
0.025	average queue size	0.024999	0.028690	0.028177	0.026864	0.0143267
	average power consumption	0.962842	0.962732	0.962747	0.962786	0.9631947
0.05	average queue size	0.049999	0.060591	0.059150	0.055350	0.0170936
	average power consumption	0.961950	0.961641	0.961683	0.961794	0.9629424
0.1	average queue size	0.099999	0.124273	0.120996	0.112271	0.0226561
	average power consumption	0.960167	0.959464	0.959559	0.959812	0.9624347
0.15	average queue size	0.149999	0.187799	0.182711	0.169121	0.0282572
	average power consumption	0.958384	0.957293	0.957439	0.957832	0.9619231
0.2	average queue size	0.199999	0.251169	0.244294	0.225903	0.0338971
	average power consumption	0.956600	0.955127	0.955324	0.955855	0.9614075
0.25	average queue size	0.249999	0.314383	0.305746	0.282615	0.0395764
	average power consumption	0.954817	0.952966	0.953214	0.953880	0.9608878
0.5	average queue size	0.499999	0.628138	0.611060	0.565140	0.0685773
	average power consumption	0.945901	0.942241	0.942728	0.944041	0.9582275
1	average queue size	0.999999	1.244302	1.212106	1.125065	0.1297798
	average power consumption	0.928068	0.921180	0.922088	0.924544	0.9525747
2	average queue size	1.999999	2.433107	2.377250	2.224843	0.2667661
	average power consumption	0.892403	0.880553	0.882081	0.886254	0.9397238
3	average queue size	2.999999	3.567222	3.495604	3.298647	0.4273874
	average power consumption	0.856737	0.841806	0.843691	0.848880	0.9242789
4	average queue size	3.999999	4.650334	4.569931	4.347387	0.6183381
	average power consumption	0.821072	0.804817	0.806827	0.812395	0.9053298
5	average queue size	4.999999	5.685810	5.602779	5.371931	0.8491042
	average power consumption	0.785406	0.769478	0.771407	0.776775	0.8814583
10	average queue size	9.999999	10.247728	10.221527	10.159411	3.5106174
	average power consumption	0.607079	0.616149	0.615189	0.612903	0.3721799
20	average queue size	10.125542	10.350669	10.327055	10.272907	3.6541356
	average power consumption	0.602601	0.613076	0.611976	0.609440	0.3052502

Table 5. Performance of Different Stochastic Policies under various Inter-arrival Distributions: Computed using PRISM.

[11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. TOOLS'02*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[12] Microsoft. OnNow Power Management Architecture for Applications. Website, August 1997.

[13] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Proceedings of Design Automation Conference*, 1998.

[14] C. Pereira, R. Gupta, P. Spanos, and M. Srivastava. A Power Aware API. *Power Aware Computing*, 2002.

[15] PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism/`.

[16] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of Design Automation Conference*, pages 555–561, June 1999.

[17] Q. Qiu and Q. Wu and M. Pedram. Stochastic Modeling of a Power-Managed System: Construction and Optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999.

[18] Q. Qiu and Q. Wu and M. Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *Proceedings of Design Automation Conference*, pages 352–356, June 2000.

[19] D. Ramanathan, S. Irani, and R. Gupta. An Analysis of System Level Power Management Algorithms and their effects on Latency. *IEEE Trans. on Computer Aided Design*, 21(3), march 2002.

[20] D. Ramanathan, S. Irani, and R. K. Gupta. Latency Effects of System Level Power Management Algorithms. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, 2000.

[21] S. Irani and S. Shukla and R. Gupta. Competitive analysis of dynamic power management strategies for systems with multiple power saving states. In *Proceedings of the Design Automation and Test Europe Conference*, 2002.

[22] S. Shukla and R. Gupta. A Model Checking Approach to Evaluating System Level Power Management for Embedded Systems. In *Proceedings of IEEE Workshop on High Level Design Validation and Test (HLDVT01)*. IEEE Press, November 2001.

[23] T. Simunic, L. Benini, and G. D. Micheli. Event Driven Power Management of Portable Systems. In *In the Proceedings of International Symposium on System Synthesis*, pages 18–23, 1999.

[24] M. B. Srivastava, A. P. Chandrakasan, and R. W. Broderson. Predictive Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Trans. on VLSI Systems*, 4(1):42–54, march 1996.