System-level Dynamic Power Management

L. Benini A. Bogliolo G. De Micheli †

DEIS - University of Bologna
Bologna, ITALY 40136

† CSL - Stanford University
Stanford, CA 94305

Abstract

We introduce the design methodology known as dynamic power management (DPM), targeting the maximization of power efficiency under performance constraints for electronic systems. We first describe the basic motivations for implementing DPM, then we survey several power management schemes. Finally, we provide guidelines to assessing the potential impact of a DPM scheme for a given target system.

1 Introduction

Electronic circuits and systems are usually designed to deliver peak performance, but in many cases peak performance levels are not needed for most of the operation time. Cellular phones and portable computers are just two examples of systems with non-uniform workload. When the user is making or receiving a call with a cellular phone (or he/she is compiling a C program with a PC), he/she wants to have maximum performance. However, when the user is carrying the phone in his pocket (or he/she is thinking to what to write next during a text-editing session on a PC), he/she does not need the full computational power of the system.

On the other hand, an increasingly large class of electronic systems is designed with power consumption in mind. Portable systems are obviously highly power-constrained, in an effort to extend battery life and decrease battery weight. Even stationary equipment is power constrained, due to increasing concerns about the cost and noise of cooling systems, the cost of electric power (for large systems) and stricter environmental impact regulations.

Dynamic power management (DPM) reduces power consumption by dynamically adjusting performance levels to the workload. A general circuit or system can be seen as a collection of interacting *resources*. If workload is non-stationary, some (or many) resources may be idle when the environment does not maximally load the system. The basic idea in DPM is to put underutilized or idle resources into states of operation with reduced or null performance levels that require little or no power.

We call *power manager* (PM) the system component (hardware or software) that performs DPM. A power manager consists of an *observer* that monitors the load of the system and its resources, a *controller*, that issues commands for forcing state transitions in system resources, and a *policy*, which is a control algorithm for deciding when and how to force state transitions (based on information provided by the observer). The two main issues in implementing DPM are: how to design a system that can be efficiently managed with minimal power waste and reduced performance penalty, and how to implement the power manager. Regarding PM implementation, we face three main challenges: controller design, observer design and policy optimization.

In the rest of the paper, we will first review how systems have been designed to enable efficient DPM. Then, we will focus on PM design and, more specifically, on policy optimization. We will survey several classes of policies and analyze how and when they can be successfully applied.

0-7695-0019-6/99 $10.00 © 1999 IEEE

2 Power Manageable Systems

In this section, we will take a bottom-up view. We will first focus on how power-manageable system components (e.g., chips) have been designed. Then we will see what is done for enabling DPM for functionally complete systems containing many interacting components. Finally, we analyze the problem of managing power for a collection of communicating systems (a *network*).

2.1 Component design

Our working definition of *component* is general. A component is a hardware block implementing a task in a complete system. Notice that the granularity of this definition is arbitrary, hence components can be as simple as a functional unit within a chip, or as complex as a board. We will first focus on power management for single chips and their components.

A large body of research has been dedicated to chip-level power management (see [1] for a survey). One of the most common techniques at the chip level is *clock gating*: whenever a functional unit becomes idle, its clock signal is stopped, preventing power consumption caused by spurious switching. Many commercial microprocessors implement clock gating to reduce power dissipation [7, 8, 9, 10]. Even though clock gating often requires handcrafted implementations, several algorithmic techniques have been developed for automatically synthesizing clock-gating logic [2, 3, 4, 5, 6].

Clock gating is a successful PM technique mainly because it is possible to resume normal system activity in a very short time (i.e., one or a few clock periods). The main challenge in implementing clock gating is efficient idleness detection. Idleness detection logic can be added to a design if its power consumption is much smaller than the power saved. In some microprocessors [7], units have dedicated idleness detection logic that stops their clock without any external control. The clock distribution for the entire microprocessor can be gated, but this decision is not taken autonomously by the microprocessor itself: special instructions are provided to force global clock freezing.

It is important to notice that clock gating does not completely eliminate power dissipation. First, even if the clock is not distributed, clock-generation circuitry is still active and dissipates power. Second, the chip still dissipates leakage power, which is not reduced by clock gating, Microprocessors and chips for battery-operated systems have stringent quiescent power constraints that may not be met by clock gating. Hence, more aggressive power-down techniques have been implemented.

In order to completely eliminate power, it is possible to: (i) disable the clock-generation circuitry, (ii) completely turn off the power supply to parts of the chip. Both these techniques imply non-negligible delay for returning to the active state. In some cases, activation times are several orders of magnitude longer than the time required for clock gating [9].

In general, we can model a power manageable component as a finite-state process. States are the various mode of operation that span the tradeoff between performance and power. State transitions have a power and delay cost. In general, low power states have lower performance and larger latency than states with higher power. This simple abstract model holds for many devices such as disk drives [11], wireless network interfaces [12], displays [11], that are more heterogeneous and complex than a single chip.

2.2 System design

From our viewpoint, a system is a set of interacting components that implement a given specification. Notice that this generic definition does not pose any limitation on the size and complexity of a system. The activity of components is coordinated by a system controller. For complex systems, control is often implemented in software. For instance, in computer systems, global coordination is performed by the *operating system* (OS).

The OS has precise and up-to-date information on the workload and on the status of system resources, hence the power manager is naturally implemented as a module of the OS. This observation is the basis of the OnNow initiative promoted by Microsoft corporation [13], that fosters *operating system directed power management* (OSPM). OnNow specifies some requirements for computer systems, namely: (i) the turn-on delay should be negligible, the computer appears to be off,

but it is capable of responding to wake-up calls; (ii) the OS controls power and performance levels; (iii) applications are aware of the possibility of finding hardware resources in different service rates; (iv) resources participate in DPM by responding to OS commands. *OnNow* is clearly not the only effort in designing power manageable computer systems controlled by software: many hardware vendors have developed software drivers that support DPM, and several researchers are investigating power-efficient software design techniques [14].

The design of systems that can be managed via software (through the OS) is also supported by an open interface specification called *advanced configuration and power management interface* (ACPI), promoted by Intel, Microsoft and Toshiba [15]. ACPI supports a finite-state model for system resources, and specifies the hardware/software interface that should be used to control them. The objectives of ACPI are to enable dynamic re-configuration and to support OSPM. It is important to notice that ACPI just specifies the interface between hardware and software and does not pose constraints on how the OS should take power management decisions, or how hardware vendors should implement component with multiple power states.

Although ACPI is primarily dedicated to computers, it can be used as a reference for a more general class of electronic systems. A power-manageable system should provide clean finite-strate abstraction of its components to the power manager and should be able to provide information on workload and resource usage. Standardization of the interface between PM and system is an important feature for decreasing design time.

2.3 Network design

In many cases, systems are not isolated, but they actively communicate among themselves. We call *network* a set of communicating systems. While network design has been traditionally focused on communication quality and throughput, the increased emphasis on low-power portable systems with communication capabilities has spurred several research initiatives targeting power-efficient networking [16].

Energy-conscious communication protocols based on power management have been extensively studied [17, 18, 19]. The main purpose of these protocols is to regulate the access of several communication devices to a shared medium trying to obtain maximum power efficiency for a given throughput requirement. Even when interference is not an issue, point-to-point communication can be made more power efficient by increasing the predictability of communication patterns [20]: if it is possible to accurately predict the arrival time of messages (packets), idle times can be exploited to force communication devices into a low-power inactive state.

3 Power manager design

In this section we will analyze techniques for controlling the power state of a system and its components. We assume that the system implements the interface layers required for supporting the finite-state abstraction described in the previous section. We focus on how to design effective power management policies, under the assumption that control and observation tools are provided within the system. Even though coordination of DPM for multiple resources and systems is an interesting open issue, for the sake of simplicity, we will focus on controlling a single resource (or, equivalently, the state of a system as a whole).

First, it is important to clarify why DPM is a non-trivial problem. Consider a system where transition between power states are instantaneous: negligible power and performance costs would be paid for peforming state transitions. In such a system, DPM is a trivial task, and the optimum policy is greedy: as soon as the system is idle, it can be transitioned to the deepest sleep state available. On the arrival of a request, the system is instantaneously activated. Unfortunately, most power manageable systems have non-negligible performance and power cost for power state transitions. For instance, if entering a low-power state requires power-supply shutdown, returning from this state to the active state requires a (possibly long) time for turning on and stabilizing the power supply. If power state transitions have a cost, we are faced with a non-trivial optimization

problem: we need to decide when it is worthwhile (performance and power-wise) to transition to a low-power state, and which state should be chosen (if multiple low-power states are available).

Consider a simple system with two power states: on and `off`. Power in the on state is P_{on} and the system is fully operational. Power in the `off` state is P_{off} and performance is null. Power for a on to `off` transition is $P_{on,off}$ and the time required is $\Delta T_{on,off}$. The opposite transition has $P_{off,on}$ and $\Delta T_{on,off}$ power and performance costs, respectively. The costs for going to the `off` state and coming back to the active state are: $P_{sd} = P_{on,off} + P_{off,on}$ and $T_{sd} = \Delta T_{on,off} + \Delta T_{off,on}$. If we decide to go to the `off` state, the power savings should be enough to compensate for the cost incurred in state transitions. Furthermore, the performance penalty should be lower than a given constraint. Notice that the DPM problem becomes trivial if there are no performance constraints: we could keep the system always `off`.

From this introductive analysis, we conclude that policy optimization is a *power optimization* problem under *performance constraints*. In the next subsections, we survey two different approaches to policy optimization, namely *predictive techniques* and *stochastic control*. To illustrate the basic techniques we will use the simple two-state system described above.

3.1 Predictive Techniques

The rationale in all predictive approaches is to take DPM decisions based on predictions on the duration of idle periods. A generic predictive method observes the time-varying workload, and, based on this observation, computes a predicted duration T_{pred} of the upcoming idle time. The PM then decides to transition to the `off` state if $T_{pred} \geq T_{BE}$, where T_{BE} is the *break-even* time, the minimum idle time long enough to amortize the state transition cost.

Good predictive approaches should minimize the time in which the system wastes power because the predictor does not signal the beginning of an idle period. Second, they should minimize the mis-predictions $T_{pred} \neq T_{idle}$, where T_{idle} is the actual duration of an idle period. We call *over-prediction* the case $T_{pred} > T_{idle}$: over-predictions always have performance and power cost. We call *under-prediction* the case $T_{pred} < T_{idle}$: under-predictions imply power waste, but may not incur performance costs.

The most common predictive PM policy is the *fixed timeout*. The policy can be summarized as follows: when an idle period begins, a timer is started with duration T_{TO}. If after T_{TO} the system is still idle, then the PM forces the transition to the `off` state. The system remains `off` until receives a request from the environment that signals the end of the idle period. The fundamental assumption in the fixed time-out policy is that the probability of T_{idle} being longer than $T_{BE} + T_{TO}$, given that $T_{idle} > T_{TO}$ is close to one: $Pr(T_{idle} > T_{TO} + T_{BE} | T_{idle} > T_{TO}) \approx 1$. Time-outs are "implicitly" predictive even if they never generate an actual T_{pred}, in the sense that they predict a long idle time if the system has been idle for a while.

The critical design decision is obviously the choice of the time-out value T_{TO}. In the next section we will analyze the tradeoffs involved in such choice. The main limitations in fixed time-outs are: (i) they may be ineffective when workload is non-stationary, and some form of adaptation is required; (ii) power is wasted while waiting for the timeout to expire; (iii) performance penalty is always paid upon wakeup.

To address the first limitation, several adaptive time-out policies have been proposed: in [21] a set of time-out values is maintained and each time-out is associated with an index indicating how successful it would have been. The policy chooses, at each idle time, the time-out that would have performed best among the set of available ones. Another policy, presented in [22], also keeps a list of candidate time-outs, and assigns a weight to each time-out based on how well it would have performed relatively to an optimum off-line strategy for past requests. The actual time-out is obtained as a weighted average of all candidates with their weights. Another approach [23] is to keep only one time-out value and to increase it when it is causing too many shutdowns. The time-out is decrease when more shutdowns can be tolerated. Several predictive policies are surveyed and compared in [24], where a general classification scheme is also proposed.

Predictive shut-down policies [25] address the second limitation of time-outs, namely the power wasted while waiting for the timeout to expire. These policies are aggressive and shut down a system

as soon as it becomes idle, if they predict $T_{pred} > T_{BE}$. A prediction of idle time duration is made available as soon as the idle period begins. Predictions are made based on past history. Past idle periods T_{idle}^{n-i} and busy periods T_{active}^{n-i} are observed to build the prediction for T_{idle}^{n}.

Two approaches have been proposed in [25]. In the first, a non-linear regression equation is obtained from past history to predict idle time: $T_{pred} = \phi(T_{active}^{n}, T_{idle}^{n-1}, \ldots, T_{active}^{n-k}, T_{idle}^{n-k+1})$. If $T_{pred} > T_{BE}$ the system is immediately shut down as soon as it becomes idle. The system wakes up only upon arrival of a service request from the environment. The format of the non-linear regression is decided heuristically, by a human designer, while the fitting coefficients can be computed with standard techniques. The main limitations of this approach are: (i) there is no automatic way to decide the type of regression equation; (ii) off-line data collection and analysis are required to construct and fit the regression model.

The second policy proposed in [25] is based on a *threshold*. The duration of the busy period T_{active}^{n} immediately preceding the current idle period is observed. If $T_{active} < T_{Thr}$, the T_{pred} is taken to be larger than T_{BE} and the system is shut down. The rationale of this policy is that for the class of systems considered in [25] (interactive graphic terminals), short active periods are often followed by long idle periods. Clearly, the choice of T_{Thr} is critical. Careful analysis of the scatterplot is required to set it to a correct value, hence this method is inherently off-line (i.e., based on extensive data collection and analysis). Furthermore, the method is not applicable if the scatterplot is not L-shaped.

Another aggressive shutdown policy has been proposed in [26]. This policy is capable of on-line adaptation, since the predicted idle time T_{pred}^{n} is obtained as a weighted sum of the last idle period T_{idle}^{n-1} and the last prediction T_{pred}^{n-1}: $T_{pred}^{n} = aT_{idle}^{n-1} + (1-a)T_{pred}^{n-1}$. Under-prediction impact is mitigated by employing a time-out scheme to re-evaluate T_{pred} periodically if the system is idle and it has not been shut down. Over-prediction impact is reduced by imposing a saturation condition on predictions: $T_{pred}^{n} < C_{max}T_{pred}^{n-1}$. The policy proposed in [26] also addresses the third limitation of time-out policies, namely the performance penalty that is always paid on wakeup. To reduce this cost, the policy performs *predictive wakeup* when the predicted idle time expires, even if no new requests have arrived. This choice may increase power dissipation if T_{idle} has been under-predicted, but decreases the delay for servicing the first incoming request after an idle period.

3.2 Stochastic control

Policy optimization is an optimization problem under uncertainty. Predictive approaches address workload uncertainty, but they assume deterministic response and transition times for the system. However, the system model for policy optimization is very abstract, and abstraction introduces uncertainty. Hence, it may be safer, and more general, to assume a stochastic model for the system as well. Second, predictive algorithms are based on a two-state system model, while real-life systems have multiple power states. Policy optimization involves not only the choice of *when* to perform state transitions, but also the choice of *which* transition should be performed. Third, predictive algorithms are heuristic, and their optimality can only be gauged through comparative simulation. Parameter tuning for these algorithms can be very hard if many parameters are involved. Finally, predictive algorithms are geared toward power minimization, and cannot finely control performance penalty.

The stochastic control approach addresses the generality and optimality issues outlined above. Rather than trying to eliminate uncertainty by prediction, it formulates policy optimization as an optimization problem under uncertainty. More specifically [27], power management optimization has been studied within the framework of *controlled Markov processes* [28, 29]. In this flavor of stochastic optimization it is assumed that the system and the workload can be modeled as Markov chains. Under this assumption, it is possible to: (i) model the uncertainty in system power consumption and response (transition) times; (ii) model complex systems with many power states, buffers, queues, etc.; (iii) compute power management policies that are globally optimum; (iv) explore tradeoffs between power and performance in a controlled fashion. The Markov model postulated by the stochastic control approach [27] consists of:

- A *service requester* (SR), a Markov chain with state set R, that models the arrival of service requests for the system (i.e, the workload).
- A *service provider* (SP), a Controlled Markov chain with S states that models the system. Its states represent the modes of operation of the system (i.e, its power states), its transitions are probabilistic, and probabilities are controlled by commands issued by the power manager.
- A *power manager* (PM), that implements a function $f : S \times R \rightarrow A$ from the state set of SR and SP to the set of possible commands A. Such function is an abstract representation of a decision process: the PM observes the state of the system and the workload, takes a decision and issues a command to control the future state of the system.
- Cost metrics, that associate power and performance values to each system state-command pair in $S \times A$.

In [27], the general Markov model is specialized by assuming finite state set, finite command set and discrete (or slotted) time. To perform policy optimization, the Markov chains of SR and SP are composed to obtain a global controlled Markov chain. Then, the problem of finding a minimum-power policy that meets given performance constraints can be cast as a linear program (LP). The solution of the LP produces a *stationary, randomized* policy. Such a policy is a non-deterministic function which, given a present system state, associates a probability to each command. The command to be issued is selected by a random trial based on the state-dependent probabilities. It can be shown [29] that the policy computed by LP is *globally optimum*. Furthermore, LP can be solved in polynomial time in the number of variables. Hence, policy optimization for Markov processes is exact and computationally efficient.

The stochastic optimization approach enjoys desirable properties of flexibility, global optimality and mathematical soundness. However, several important points need to be understood. First, the performance and power obtained by a policy are *expected* values, there is no guarantee that results will be optimum for a specific workload instance (i.e., a single realization of the corresponding stochastic process). Second, policy optimization requires a Markov model for SP and SR. If we can safely assume that the SP model can be pre-characterized, we cannot assume that we always know the SR model beforehand. Third, policy implementation in practice may not be straightforward. We have always implicitly assumed that the power consumption of the PM is negligible, but this assumption needs to be validated on a case-by-case basis. Finally, the Markov model for the SR or SP can be just an approximation of a much more complex stochastic process. If the model is not accurate, then the "optimal" policies are just approximate solutions.

4 Analysis of power-managed systems

The potential impact of applying DPM to a given system depends on system parameters and workload statistics. In this section we study the joint effect of system and workload parameters in order to evaluate the suitability of DPM and provide guidelines for selecting among power management schemes.

We divide this section into three parts: in the first part we discuss the general applicability of DPM by referring to an ideal power manager, while in the second and third parts we focus on predictive techniques and stochastic control, respectively.

4.1 Applicability of DMP

Consider a simple resource with only one sleep state controlled by an *ideal* PM having complete (a priori) knowledge of the entire workload trace. The optimum policy for the ideal PM consists of shutting down the resource at the beginning af all idle periods longer than the break-even time (i.e., long enough to compensate the cost of shut-down) and waking it up right in time to serve upcoming requests with no delay. The resulting power consumption (P_{ideal}) is a lower bound for the power consumption that can be achieved by means of DPM without impairing performance.

We denote by P_{saved} the potential power saving (*i.e.*, the gap between P_{ideal} and the power consumption of the active state, P_{on}) and we study its sensitivity to system parameters and workload statistics. System parameters are represented by the break-even time T_{BE}, workload statistics are represented by the probability density function (pdf) of idle periods T_{idle}. Intuitively, the larger T_{BE} (with respect to the average idle time) the smaller P_{saved}. In the limiting situation where all idle periods are shorter than T_{BE}, no power savings would be achieved by means of DPM: an ideal PM implementing the optimum policy would never shut the resource down, thus providing $P_{ideal} = P_{on}$ and $P_{saved} = 0$.

For a two-state component, T_{BE} can be expressed as a function of the shut-down cost and delay (P_{sd} and T_{sd}, as defined in Section 3) and of the power consumption in on and `off` states (P_{on} and P_{off}):

$$T_{BE} = T_{sd} + T_{sd}\frac{P_{sd} - P_{on}}{P_{on} - P_{off}} \tag{1}$$

In practice, T_{BE} grows linearly with T_{sd} and P_{sd}, and depends hyperbolically on $P_{on} - P_{off}$. For systems with multiple sleep states, a different T_{BE} has to be defined for each state. In general, deeper sleep states have lower power consumption at the cost of longer and more expensive transitions. When designing power-manageable components, a trade-off between P_{off}, P_{sd} and T_{sd} has to be found for each sleep state to obtain small T_{BE}. Sleep states with smaller T_{BE} are more likely to be successfully exploited by DPM.

If we denote by F the probability distribution of the idle periods, and by $T^{avg}_{idle>T_{BE}}$ the average length of idle periods longer than T_{BE}, P_{saved} can be expressed as the product of three terms: the power saving of the sleep state, the expected idle time in excess of T_{BE} and the probability of going to sleep according to the optimum ideal policy.

$$P_{saved}(T_{BE}) = (P_{on} - P_{off})(T^{avg}_{idle>T_{BE}} - T_{BE})(1 - F(T_{BE})) \tag{2}$$

P_{saved} is always a decreasing function of T_{BE}: it takes maximum value for $T_{BE} = 0$ and asymptotically tends to 0 for increasing values of T_{BE}. The way it goes to zero depends on the first order statistics of the workload, namely, on the distribution of T_{idle}.

As a rule of thumb, we can say that DPM can be successfully applyed to a power manageable system whenever the T_{BE} of (some of) its sleep states is smaller than the average idle time of the workload.

4.2 Predictive techniques

In most real-world systems there is no knowledge of future input events and DPM decisions have to be taken based on uncertain predictions. Predictors exploit the correlation between the future event to be predicted and the past history of the workload. We denote by p and o, respectively, the event to be predicted and the observed event the prediction is based on. We are interested in predicting idle periods long enough to go to sleep, in symbols: $p = \{T_{idle} > T_{BE}\}$.

We define two figures to represent the quality of a predictor: ***safety***, that is the complement of the risk of making wrong predictions, and ***efficiency***, that is the complement of the risk of missing a prediction. Safety and efficiency can be expressed in terms of conditional probabilities $Prob(p|o)$ and $Prob(o|p)$, respectively. A safe predictor never makes overpredictions ($Prob(p|o) = 1$), an efficient predictor never makes underpredictions ($Prob(o|p) = 1$). A predictor with maximum safety and efficiency is an *ideal predictor*, whose availability would enable the actual implementation of the ideal PM discussed in the previous subsection. Predictors of practical interest are neither safe nor efficient, thus causing sub-optimum control. Their quality (and the quality of the resulting control) depends on the choice of the observed event o and on the second-order workload statistics.

Predictors based on timeouts use the elapsed idle time as observed event: $o = \{T^{a}_{idle} > T_{TO}\}$. Timeouts have two main advantages: they are general (their applicability slightly depends on the workload) and they are safe (safety can be improved simply by increasing the timeout value). Unfortunately, they trade-off efficiency for safety: large timeouts cause a large number of underpredictions, and a sizeable amount of power is wasted (during user's idleness) waiting for the timeout to expire.

This last issue is addressed by predictors based on the past history [24, 25], that observe the length of the last idle/active periods: e.g., $o = \{T_{active}^{n-1} < T_{th}\}$. The length of the next idle period is predicted a priori, thus enabling the PM to take a decision as soon as a new idle period starts. On the other hand, the applicability and the quality of history-based predictors depend on the correlation between past and future events, that is not under designer's control. As a matter of fact, short-term correlation has been observed in many real-world workloads, but the nature and strength of such correlation is strongly instance dependent. For a given workload, history-based predictors are usually more efficient and less safe than time-outs.

For non-stationary workloads adaptive predictors are required [21, 22, 23, 26]. While for timeouts the only parameter to be adjusted is the timer duration, for history-based predictors even the type of observed events should in principle be adapted to the workload.

4.3 Stochastic control

Stochastic control based on Markov models has several advantages over predictive techniques. First, it captures the global view of the system, thus allowing the designer to search for a global optimum that possibly exploits multiple inactive states of multiple interacting resources. Second, it enables the exact solution (in polynomial time) of the performance-constrained power optimization problem. Third, it exploits the strength and optimality of randomized policies.

The scope of application of stochastic control has two limits: modeling assumptions and complexity. If the system or the workload are non Markovian, optimality cannot be proved and the expected results need to be validated against simulation, emulation or implementation. Second, implementing a randomized policy for stochastic control is usually more complex than implementing predictive techniques. While complexity is usually not an issue for PMs implemented as software components, it may be a concern for hardware PMs.

The ideal application of stochastic control are computer-based systems subject to performance constraints. We remark, however, that policy optimization can be used as a powerful tool for design exploration even when stochastic control is not the target DPM technique. In fact, once Markov models have been constructed for the system and the workload, the Pareto curve of optimum tradeoff points can be drawn on the power-performance plane by repeatedly solving policy optimization while varying performance constraints. The Pareto curve provides valuable information to evaluate and improve the quality of any power management strategy.

5 Conclusions

DPM exploits variations in workload to save power. We have surveyed Several DPM techniques and discussed their potential and applicability. We have shown that DPM can be effectively applyed to a variety of systems to provide sizable power savings with small performance penalties.

Most systems today use some form of DPM, but they are often based on ad-hoc heuristics that are far from the global optimum and cannot be extended to deal with the growing number of degrees of freedom made available by today's power-manageable components. On the other hand, we are only at the beginning of the development of advanced power management schemes, whose design complexity is prompting for the development of new CAD tools.

References

[1] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*, Kluwer, 1997.

[2] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *IEEE Transactions on VLSI Systems*, Vol. VLSI-2, no. 4, pp. 426-436, Dec. 1994.

[3] S. malik V. Tiwari and P. Ashar, "Guarded evaluation: Pushing Power Management to Logic Synthesis/Design," in *International Symposium on Low Power Design*, pp. 221–226, Apr. 1995.

[4] L. Benini and G. De Micheli, "Transformation and Synthesis of FSMs for Low Power Gated Clock Implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, no. 6, pp. 630-643, June 1996.

[5] F. Theeuwen and E. Seelen, "Power Reduction Through Clock Gating by Symbolic Manipulation," in *Symposium on Logic and Architecture Design*, pp. 184–191, Dec. 1996.

[6] M. Ohnishi et al., "A method of Redundant Clocking Detection and Power Reduction at RT-level Design," in *International Symposium on Low Power Design*, pp. 131–136, Aug. 1997.

[7] S. Gary, P. Ippolito et al., "PowerPC 603, a Microprocessor for Portable Computers," *IEEE Design & Test of Computers* vol. 11, no. 4, pp. 14-23, Win. 1994.

[8] G. Debnath, K. Debnath and R. Fernando, "The Pentium Processor-90/100, Microarchitecture and Low-Power Circuit Design," in *International conference on VLSI design*, pp. 185–190, Jan. 1995.

[9] S. Furber, *ARM System Architecture* Addison-Wesley, 1997.

[10] M. Gowan, L. Biro and D. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," in *Design Automation Conference*, pp. 726-731, June 1998.

[11] E. Harris et al. "Technology Directions for Portable Computers," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 636–657, April 1996.

[12] M. Stemm and R. Katz, "Measuring and Reducing Energy Consumption of Network Interfaces in Hand-held Devices", *IEICE Transactions on Communications*, vol. E80-B, no. 8, pp. 1125–1131, Aug. 1997.

[13] Microsoft, "OnNow: the Evolution of the PC Platform", available at *http://www.microsoft.com/hwdev/pcfuture/ONNOW.HTM*, 1997.

[14] J. Lorch and A. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Communications*, vol. 5, no. 3, June 1998.

[15] Intel, Microsoft and Toshiba, "Advanced Configuration and Power Interface Specification", available at *http://www.intel.com/ial/powermgm/specs.html*, 1996.

[16] N. Bambos, "Toward Power-Sensitive Network Architectures in Wireless Communications: Concepts, Issues and Design Aspects," *IEEE Personal Communications*, vol. 5, no. 3, pp. 50–59, June 1998.

[17] J. Rulnick and N. Bambos, "Mobile Power Management for Wireless Communication Networks," *Wireless Networks*, vol. 3, no. 1, pp. 3–14, Jan. 1997.

[18] K. Sivalingham, M. Srivastava et al., "Low-power Access Protocols Based on Scheduling for Wireless and Mobile ATM Networks," in *Int.l Conference on Universal Personal Communications*, pp. 429–433, 1997.

[19] M. Zorzi and R. Rao, "Energy-Constrained Error Control for Wireless Channels," *IEEE Personal Communications*, vol. 4, no. 6, pp. 27–33, Dec. 1997.

[20] B. Mangione-Smith, "Low-Power Communication Protocols: Paging and Beyond," *IEEE Symposium on Low-Power Electronics*, pp. 8–11, 1995.

[21] P. Krishnan, P. Long and J. Vitter, "Adaptive Disk Spindown Via Optimal Rent-to-buy in Probabilistic Environments," in *Int'l Conference on Machine Learning*, pp. 322-330, July 1995.

[22] D. Helmbold, D. Long, E. Sherrod, "Dynamic Disk Spin-down Technique for Mobile Computing," in *IEEE Conference on Mobile Computing*, pp. 130–142, Nov. 1996.

[23] F. Douglis, P. Krishnan and B. Bershad, "Adaptive Disk Spin-down Policies for Mobile Computers," in *Second USENIX Symposium on Mobile and Location-Independent Computing*, pp. 121–137, Apr. 1995.

[24] R Golding, P. Bosh and J. Wilkes, "Idleness is not Sloth" *HP Laboratories Technical Report* HPL-96-140, 1996.

[25] M. Srivastava, A. Chandrakasan. R. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 42–55, March 1996.

[26] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", in *Int.l Conference on Computer Aided Design*, pp. 28-32, Nov. 1997.

[27] G. Paleologo, L. Benini, A. Bogliolo and G. De Micheli, "Policy Optimization for Dynamic Power Management," in *Design Automation Conference*, pp. 182-187, June 1998.

[28] S. Ross, *Introduction to Probability models*, 6^{th} ed. Academic Press, 1997.

[29] M. Puterman, *Finite Markov Decision Processes*, John Wiley and Sons, 1994.