A Formal Framework for Modeling and Analysis of System-Level Dynamic Power Management

Shrirang Yardi, Karthik Channakeshava, Michael S. Hsiao, Thomas L. Martin and Dong S. Ha
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24060. USA.
{*yardi, kchannak, mhsiao, tlmartin, ha*}*@vt.edu*

Abstract

Recent advances in Dynamic Power Management (DPM) techniques have resulted in designs that support a rich set of power management options, both at the hardware and software levels. This has resulted in an explosion of the design space when analyzing the system-level tradeoffs of candidate DPM strategy designs. This paper proposes a design space exploration methodology based on a high-level, multi-layered modeling framework that facilitates rapid estimation of system-wide energy by providing the designer with a global view of the system. The framework is based on the Extended Finite State Machine formalism and abstracts the component power modes, the operating environment and the DPM architecture into interacting, concurrent layers within a single, unified model. The modeling framework is coupled with a symbolic simulation engine to allow for rapid traversal of the large design space. We first illustrate how the proposed model can be constructed by making reasonable assumptions on the system and workload parameters, and then we show how analysis of various candidate strategies can be performed using this model. Our aim is to provide a high-level model that can be used to quickly assess the impact of various power management decisions on the system-wide energy. The framework can also be a formal basis for design of energy efficient power management systems.

1 Introduction

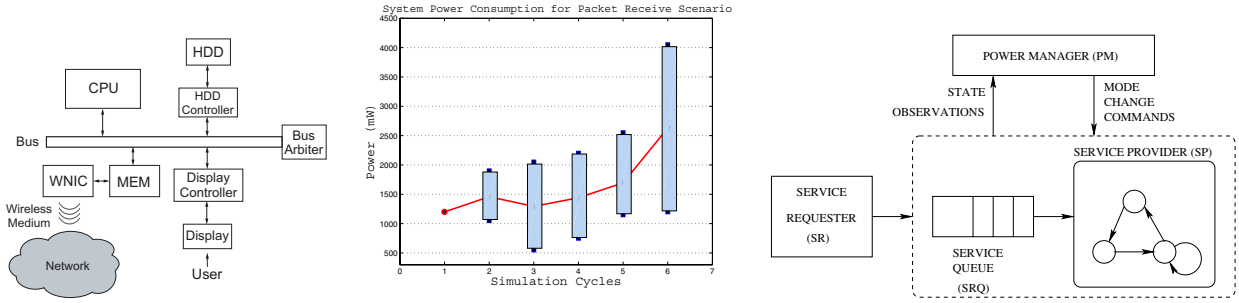
Dynamic Power Management (DPM) [3] is a system-level power reduction technique that manages available resources by selectively placing some devices in low power or idle states when there is a reduced demand for service. A typical power management system, which we term as a *DPM Architecture*, consists of a set of DPM policies and a power manager that can reside either in hardware or the operating system (OS) [4]. The job of the power manager is to control the power states of the power managed components (PMCs) such that system performance requirements are guaranteed while reducing power consumption. A thorough analysis of system-level power-performance tradeoffs during the design stage itself is, thus, an essential step to ensure an efficient DPM architecture design. Given a target architecture and a set of candidate policies, performing such an analysis at the system-level while meeting tight constraints, like time-to-market, requires efficient tools that can provide reasonably accurate estimates of the system power/performance early in the design.

Prior research in DPM has primarily focused on policy design and optimization for individual devices by considering them in isolation [4, 12, 18, 24, 25]. However, in complex systems, the interaction between different PMCs, their power mode dependencies, management decisions made at different levels and the system workload are significant factors in shaping the system power profile. This poses significant challenges when exploring system-level tradeoffs during policy design since the *power design space* (*i.e.*, the total hardware/software decisions that affect the power profile) for such systems can be very large. As system complexity increases, performing an exhaustive traversal of this design space will likely be the main bottleneck in implementing aggressive, holistic DPM architectures [2, 8, 9, 17]. In this context, there is a need of a high-level model that can provide the designer with a system-wide view to allow rapid and reasonably accurate assessment of power management decisions over this design space.

1.1 Motivation for our Modeling Framework

We motivate the need for our modeling framework by illustrating, using an example system, that existing system level models and exploration techniques used for power–performance analysis of DPM strategies are insufficient to handle the increasing system and DPM architecture complexity.

Consider the system shown in Figure 1(a) which consists of the following PMCs: a processor (CPU), hard-disk drive (HDD), display (DISPLAY) and a wireless network interface (WNIC) connected by a single, shared system bus (BUS, BUS-ARBITER). Each PMC is represented by a power state machine (PSM) [4] that models three power modes for each device – ACTIVE, IDLE and SLEEP. We want to analyze the system power profile for a single event – the WNIC receiving a packet that needs to be processed by the CPU resulting in some data to be written to the HDD. Figure 1(b) shows the min/max power consumption of the system as the packet progresses through it. The system initially starts at the all IDLE state. The WNIC is first activated to receive the packet followed by MEM where the packet is buffered resulting in the activation of the BUS-ARBITER and BUS which transfer the packet to the CPU. The CPU becomes ACTIVE to process the packet and in turn, activates the HDD-CONTROLLER and HDD where the data is finally written. Each component can go to any PSM state after it has processed the



(a) An Example System

(b) Min/Max System Power when receiving a packet at the WNIC

(c) System Model used in previous techniques for DPM policy design

Figure 1. Scenario that highlights the impact of the power mode inter-dependencies on the overall power profile

packet. PMCs that are not involved in this scenario (like the display) are assumed to be in the SLEEP state. If the initial state were different (e.g., CPU and DISPLAY were in ACTIVE) then, for the same scenario, the power profile would be significantly different.

This experiment illustrates that *in addition to the individual power behaviors, the dependencies between the PMCs and their interaction with their environment significantly impacts the overall power profile*. The primary motivation of our work is the lack of system models that can capture such behavior of complex, multi-resource, multi-tasking systems in the context of DPM and provide means for rapid exploration of their power design space. Our aim is to design a model with the following requirements:

- The model should provide a global, system-wide view by accurately capturing the inter-dependent and concurrent operation of PMCs, and their interaction with the system workload. This would allow the designer to analyze the impact of any DPM decision across the entire power design space.
- It should be both detailed enough to provide accurate estimates, and flexible enough to deal with non-determinism due to early and partial design decisions, constraints and uncertainty in the workload information.
- It should be sufficiently abstract to handle the diverse implementation possibilities of different DPM architectures. For example, DPM can be either implemented in hardware [6] or as part of the OS [8] or as part of a power-aware application [19].
- Finally, the model should support exhaustive exploration of the power design space for analysis of system-level tradeoffs.

Existing system-level models (Figure 1(c) [5, 12, 18, 25]) provide a localized, component-centric view of each PMC and hence fail to capture their complex interactions. Further, the power/performance analysis in previous approaches is typically performed using actual physical measurements [1] or simulations (trace-based [6], cycle-accurate [26] or stochastic [5, 18]). Consequently, the entire power design space may not be explored, significant opportunities for power reduction may be missed and estimation accuracy can be significantly compromised.

1.2 Overview and Contributions

To address the challenges mentioned above, we propose a modeling framework that allows efficient representation, analysis and exploration of the power design space to drive investigation of DPM policy design tradeoffs. Our framework focuses on the following aspects of the design exploration process - (i) abstract representation of PMCs and their workloads using the Extended Finite State Machine (EFSM) [16] formalism, (ii) modeling of component interactions and mode dependencies, (iii) platform independent modeling of the target DPM architecture, and (iv) efficient simulation to allow rapid enumeration of the power design space. To handle the potentially large number of states, we propose the use of symbolic simulation techniques that work with *sets of states*, specified by their characteristic functions in the form of Binary Decision Diagrams (BDDs). This avoids explicit state representation and allows use of reachability analysis techniques to exhaustively explore the design space [28].

We apply our framework to model a complex, system-level DPM architecture that jointly power-manages the hard disk and network interface of a handheld, battery-powered device. We present results of power/performance analysis of this architecture using our model and compare our estimates with extensive trace-based simulations. Our results demonstrate that such a high-level model can be an useful design aid early in the design to allow rapid exploration and analysis of complex DPM architectures.

2 Modeling Methodology

Figure 2 illustrates the overall organization of our modeling framework. The model abstracts the system components into multiple, interacting layers that capture the system power behavior at different levels. These layers are termed as–(i) the Operations Layer (OL) which models the operations performed by the system as *tasks*, (ii) the resource layer (RL) which abstracts the hardware devices as *resources* and, (iii) the Control Layer (CL) that models the DPM policies implemented by the system as a *set of rules* specifed in propositional logic. Each layer represents a specific system level by abstracting out the functionality and considering only the impact of its components on the system power profile. Entities in the OL and RL are represented as concurrent, communicating EFSMs with layer-specific state and transition semantics. Entities operate by performing timed *actions* and communicate via instantaneous *events* at each tick of a global clock signal. Events and actions are atomic and their operation granularity depends on the chosen period of the clock cycle. The period in turn, is bound by the simulation com-

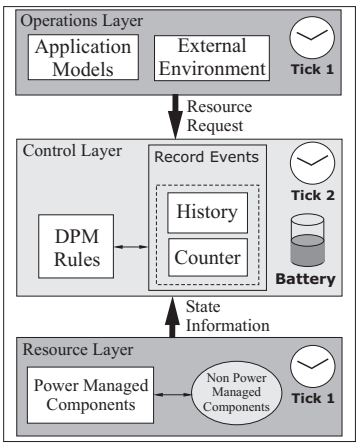


Figure 2. Multi-layered System Model–Overview

plexity and the degree of understanding of the system behavior at the abstract level. In what follows, we first explain our rationale behind the choice of such a framework, present details of each layer and demonstrate how energy and delay analysis of a target DPM architecture and its candidate policies can be efficiently performed.

2.1 Rationale

Our reasons for choosing a multi-layered, EFSM-based system representation are as follows:

- The hardware and software layers can affect the overall power profile in entirely different ways. The OL and RL layers allow the modeling of these levels using different state and transition semantics, thus providing a more accurate representation and consequently, a better understanding of the components affecting the power behavior.
- Using a separate layer to represent the DPM functions allow these to be decoupled from the functionality of the rest of the system. Hence, the monitoring and decision functionality of different policies can be represented and modified independently of the other layers. This also allows different DPM policies to be plugged in easily.
- The layered framework essentially allows decoupling of the computation and communication aspects of the target system. Each layer can be described at a different level of abstraction. This is unlike previous models where the tight coupling of the computation and communication causes system representation and simulation to quickly become intractable.
- The EFSM formalism allows an efficient mapping to BDD-based symbolic techniques which we use for design space exploration.
- Using a common formalism allows only a single concurrency model to be used for the description of the workload as well as the PMCs. This reduces design time and enhances productivity.

2.2 An Illustrative Example

In what follows, we explain details of the model components using a power managed IEEE 802.11 wireless network interface card (WNIC) operating in the infrastructure mode [14] as an illustrative example of a PMC. The WNIC can either operate in the Continuously Aware Mode (CAM) where it is always ON, or, the Power Save Mode (PSM) where it is allowed to sleep periodically. With PSM enabled, the WNIC wakes up every *Beacon Interval* (typically 100 ms), to listen to a traffic indication map (TIM) from a wired access point (AP) which buffers data for the WNIC during this interval. Once awake, the WNIC can go to sleep only when all the data from the AP has been downloaded. When the mobile device itself has any data to send, it can wake up the WNIC without waiting for the beacon. We demonstrate the steps to construct the OL-CL-RL model of this PMC and illustrate how the power/performance analysis of the PSM mode is performed using our framework.

3 Model Components

3.1 Operations Layer

The Operations Layer (OL) models the system operations or *tasks* in terms of the demands that they place on the system resources. Each task EFSM consists of finite *task states* where each state uniquely corresponds to a set of physical parameter values that define a specific system performance level. The set of all such values for a task is termed as its *operating point (OP)* [8]. State transitions are actions that request resources in the RL to transition to a new OP. The OL-RL decomposition allows the actual (physical) mechanism of such a change to be hidden from the high-level tasks. For example, a OL state transition that requests $\{\mathsf{system} = \mathsf{ACTIVE}\}$ may actually result in a mode transition sequence in the RL like $\{\mathsf{A} = \mathsf{Idle}, \mathsf{B} = \mathsf{Active}\ldots\}$. OL tasks are assumed to be incommunicado and require to explicitly request an OP to communicate with each other. This allows the model to track the effect of inter-process dependencies on the system power.

A task can have optional attributes like: (i) *foreground* which denotes a time- or system-critical task, *e.g.*, response to some sensed data, (ii) *background* which denotes a non-critical task, *e.g.*, audio playback, (iii) *privileged*– a task that can request its own OP(s), and (iv) *non-privileged*–a task for which the DPM policy selects an OP. No assumptions are made on the number or inter-arrival times of operations thus allowing completely random workloads to be modeled. Further, there are no restrictions on the interpretation of task states as long as each state corresponds to a valid operating point. To keep task EFSMs simple, a scheduler is assumed to resolve any resource conflicts between tasks.

For our example, in one possible implementation, the OL may consist of two tasks that request WNIC power mode changes–a browser that generates HTTP requests and a AP that provides the server responses back to the mobile device. The browser can be modeled as a simple two-state EFSM that represents the presence or absence of a HTTP request while the AP can consist of multiple states that indicate the fraction of the beacon interval for which the WNIC needs to be awake to receive buffered packets.

3.2 Resource Layer

The Resource Layer (RL) models the system PMCs as power state machines (PSMs). Each RL state corresponds to a precharacterized, discrete power value and state transitions represent power mode changes. Resources can be either non-power-managed (always ON), managed by the CL or be self-power-managed. State transitions may incur delay and power penalties which are modeled using additional dummy states. Communication between different PSMs represents power mode dependencies and can constrain or cause additional mode transitions. Hence, a mode transition due to an OL request for a particular resource can incur additional power

and delay due to transitions in dependent PSMs. For a given cycle, the set of all task and power mode pairs is termed as the system *operating state* [8].

No assumptions are made on the number of power modes. The low-level details about the mechanism of actual mode transitions is abstracted away. Mode dependency information can either be explicitly specified or statically derived using algorithms such as the one proposed by Li et al. [17]. Examples of resources include processors that support voltage/frequency scaling, system busses, displays, HDDs etc. On-board memories are an example of a non-power managed device. The WNIC in our example can be modeled as a PSM with SEND, RECV, BEACON and SLEEP states.

3.3 Control Layer

The Control Layer (CL) abstracts the DPM policies as a set of *rules* that are used to control the dynamic state of the system. Rules are specified as *propositional logic formulae* that operate on the OL requests and output signals that effect mode transitions in the RL. This mapping results in a set of unique task state–operating point pairs and the set of all such mappings for a given system configuration is the system operating state. If a unique mapping is not found, the CL stops the system and provides a trace of events that caused such a violation.

This design of the CL was motivated by the *Monitor-based Specification* proposed by Shimizu et al. [23] to formally specify communication protocols. This involves a monitor that is used to check the compliance of a protocol at each execution step by observing the signals output by two communicating agents. In the context of our model, the *CL can be viewed as a protocol that co-ordinates the communication between two agents (the OL and RL) according to certain rules (the DPM policy).* Further, since entities in the OL are incommunicado, our model satisfies both the *separability* and *independent implementability* [23] constraints that are essential for monitor-based specifications. This allows the CL to be modeled using *synthesizeable* logic formulae [23] which is curcial for their implementation in a real system. The only restriction on the CL specification is that it must not result in deadlock, so that for every valid state in the CL, there exist implementations of the OL and RL which cause the CL to reach that state.

The CL also monitors the system behavior using a set of state machines termed as *HISTORY* and *COUNTERS*. HISTORY is used to record a single event at some point in the past and COUNTERS can record multiple such events. For the WNIC example, these can be modeled as the BeaconCounter and TIMCounter where the former represents the beacon interval while the latter indicates the number of packets buffered for the WNIC at the AP. The CL models the PSM mode by using the OL requests (browser and AP), and these counter values, to generate the CardSleep, CardSend, CardRecv and CardBeacon signals to control the power modes of the WNIC.

3.4 System Operation

The entire system is synchronized using a global clock that is split into two parts: TICK1, on which the OL and RL are synchronized and, TICK2, during which the CL operates. On every TICK1, the OL generates requests that are processed by the CL at the following TICK2 to generate signals that effect mode transitions in the RL on the next TICK1. During this time, the CL also checks to see if all the specified constraints are satisfied. In the case of the WNIC example, the browser generates HTTP requests during TICK1 which causes the CL to issue a CardSend signal to the RL on TICK2. The WNIC then transitions to the SEND state on the next TICK1 resulting in a new operating state.

The model provides a system-wide view at each execution step by building a *System Power State Machine (SPSM)* using the synchronous composition of individual EFSMs from the OL and RL. At each execution step, all the transitions between the OL and RL are first explicitly resolved using the CL rules. This involves evaluating all the parameterized inputs and specifying each rule as a transition function for a particular task state-operating point pair. The effect of this step is to replace the CL by explicit transitions between the OL and RL resulting in one EFSM for every task state-operating point pair. Thus, for our example when the browser issues a HTTP request, the CardSend signal generated by the CL is replaced by the $\{\mathsf{HTTP\_REQ}, \mathsf{SEND}\}$ state pair. In the second step, these EFSMs are *flattened* [16] into finite state machines (termed as PSMs) each with its own transition function which is represented as a BDD. Finally, the transition function of the SPSM, termed $\mathsf{TR}_{\mathsf{sys}}$ (represented as another BDD) is obtained by composing the BDDs of individual PSMs.

Figure 3(c) illustrates the SPSM for a single execution step. Each state in the SPSM represents a combination of *all the valid task state-operating point pairs* which is nothing but the complete system operating state. An exhaustive exploration of the SPSM can then allow the designer to estimate the power consumption of the entire system for the given set of CL rules. Figure 3(b) illustrates the overall design flow of our framework.

3.5 Binding the Simulation

We use a battery model, as part of the CL, to bind the simulation time horizon of the system. A fully "charged" battery is represented by a specific number of *Time Steps* (TS) and battery charge/discharge is modeled by incrementing/decrementing a fixed value from TS every clock cycle. Rate-capacity effects are roughly modeled by varying this fixed value depending on the task attributes. For example, effects of *foreground* tasks are modeled by subtracting a higher value from TS than the value subtracted for *background* tasks. Battery re-charge can be modeled by incrementing TS during some cycles. Once TS reaches zero, further clock ticks are disabled, which stops the system.

4 Power and Delay Analysis

4.1 Power Computation Algorithm

Given the system PSM and an initial state, power computation is performed by an exhaustive traversal of the SPSM states using BDD-based symbolic simulation and adding the power for each state. We use NuSMV [10] as the input language for our model and CUDD [27] as the BDD package to perform the symbolic simulation. The default BDD encoding scheme by NuSMV is used to encode the individual PSMs and wrapper functions are used to perform the power computation. A snapshot of the NuSMV model for the WNIC example is shown in Figure 4(a).

Each execution step consists of two phases. The first corresponds to TICK1 of the clock where the task to operating point mappings are resolved. This step results in–(i) the SPSM corresponding to the CL decisions for this execution step, and (ii) a set of signals (e.g $\mathsf{A} = \mathsf{active}, \mathsf{B} = \mathsf{sleep} \ldots$) that represent these decisions. These signals are provided to the symbolic simulation algorithm which

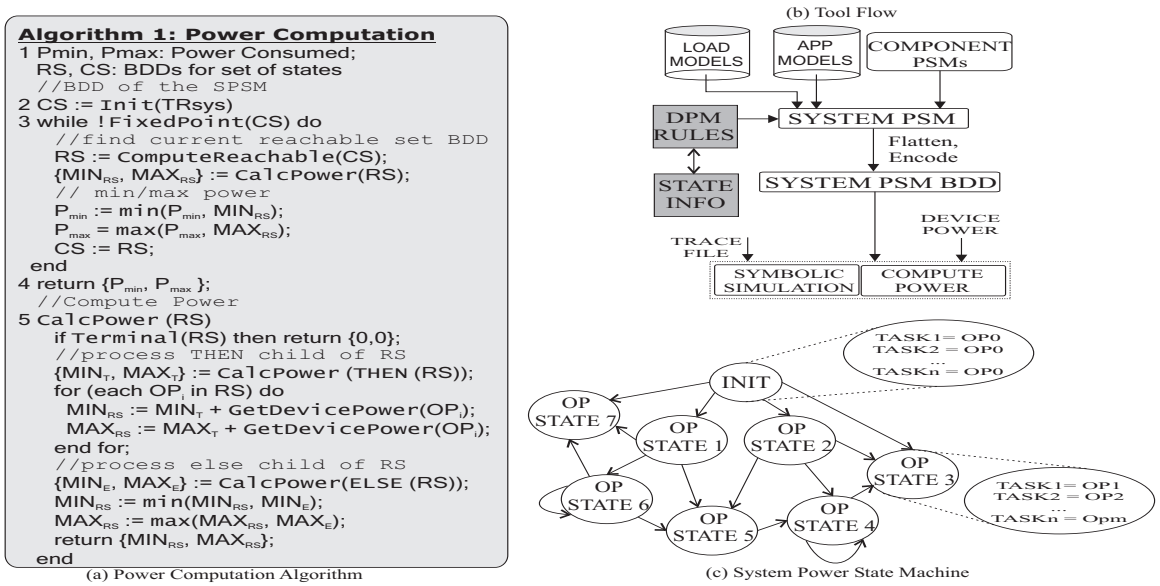
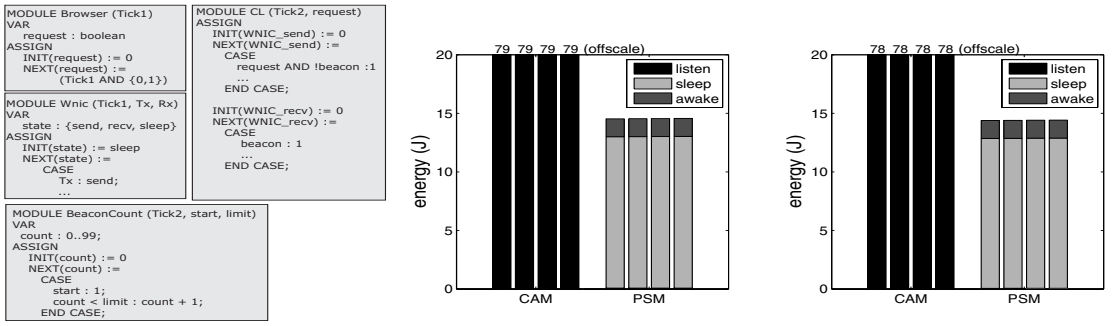


Figure 3. Power Computation, Analysis and Tool Flow using our modeling methodology



(a) The NuSMV model

(b) Energy Consumption using the NS-2 Model

(c) Energy Consumption using the OL-CL-RL Model

Figure 4. OL-CL-RL Model for a 802.11 WNIC in PSM Mode: NuSMV model and Power Analysis Results

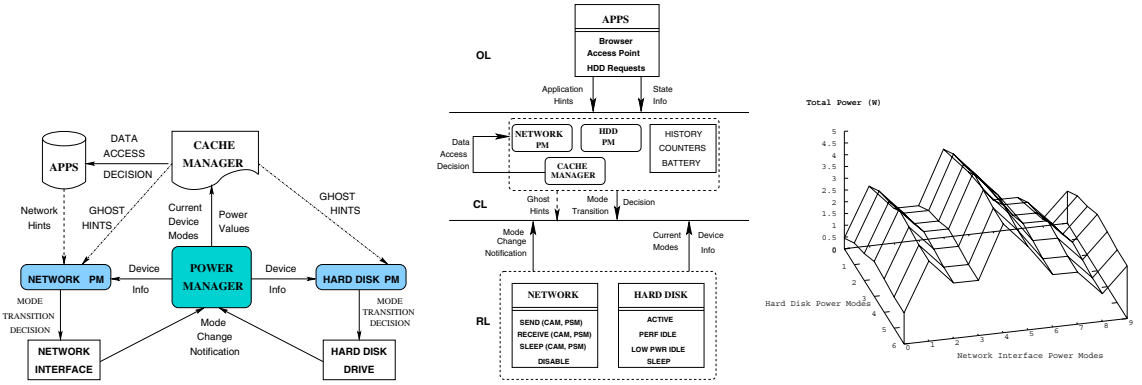
computes the reachable state space starting from the initial state of this step. Each state enumerated by the algorithm is an operating state of the system and provides a snapshot of the power behavior of the entire system. Thus, a trace of visited states can provide the impact of the CL decisions on the overall system power profile for this particular execution step. For each state, the algorithm computes the power using pre-characterized power values and reports the minimum and maximum power.

Algorithm 1 (Figure 3(a)) illustrates the steps involved in the power computation for a given set of states. This is similar to the method proposed by Bergamaschi and Jiang [7]. However, an important difference is that the signals required for successive execution steps are generated by our model itself rather than relying on real input traces. If the initial state contains any don't-cares (due to non-determinism in the workload) a state is assigned non-deterministically from the set of all possible initial states. If no system inputs are specified, the simulation is akin to formal verification, if all inputs are specified, the simulation results in a single next state.

Perfomance analysis is more challenging and highly depends on the granularity of the clock period. This is a tradeoff between estimation accuracy and simulation complexity as too small a granularity can lead to explosion of the state space. Further, since our backend tool, *i.e.* NuSMV, does not support any concept of time, we were limited in our choice of granularity to control the state space. In Section 5.5 we discuss our plans to extend our work to handle performance analysis using implicit state space traversal techniques.

4.2 Results for our Illustrative Example

The results of applying Algorithm 1 to our WNIC model are illustrated in Figure 4. We compared our estimates with energy values obtained from trace-based simulation of the PSM model designed by Krashinsky [15] for the ns-2 network simulator. Their model consists of a web-browsing mobile client, an access point and a remote server. The randomized parameters used for HTTP traffic simulation were obtained from the Berkeley Home IP traces [11]. To provide a faithful comparison, we used most of the default settings for bandwidth and latency used in [15]. Results are reported for four



(a) Example of a complex DPM architecture (b) OL-CL-RL Model (c) Power Mode Design Space for the Model

Figure 5. Modeling and Analysis of a Complex DPM Architecture using our framework

Round-Trip Time (RTT) values (5ms, 10ms, 20ms, 40ms) (Figures 4(b), 4(c)). In the worst case, the energy estimation provided by our model had an average error of 0.98% and a peak error of 16.56%. We found that this discrepancy was due to the 1 ms time resolution of atomic actions that we selected in our model which caused it to overestimate the energy values for cases where actual response times were smaller.

5 Case Study

Since our modeling framework specifies the system at a high level, the power and performance estimates provided by the model must be accurate enough for the analysis to be useful. We evaluate the model against the following criteria: (i) accuracy of the power and performance estimates when compared to simulation or measurements, (ii) how easy it is to create and modify the representation, (iii) how can larger, more complex system-level models be constructed by composing models of individual components, (iv) can the system-wide effects of power management decisions be estimated, and (v) efficiency of the framework when satisfying these criteria. We use the *Self-Tuning Power Management Architecture (STPM)* proposed by Anand et al. [1] as an example of a complex DPM architecture and use our framework to model and analyze candidate DPM policies that can be implemented within STPM.

5.1 Overview of STPM [1]

STPM is based on the observation that different DPM strategies are required for any device as their execution context changes. It is implemented as a OS module that allows device power management to adapt using information like application intent, base power of the system, time and energy costs of mode transitions and user-level power/performance preferences (termed as *knobs*). STPM provides a simple interface (via the OS) that allows applications to disclose usage *hints* to the underlying hardware so that the DPM strategy can perform decisions "intelligently" [1]. Anand et al. [2] also propose a *ghost hints interface* that allows applications to convey additional hints to devices if they were in an inappropriate power mode when it tries to access them. These hints signify "lost opportunities" on the part of the application and are used to *proactively* transition the device to the correct mode to avoid further performance penalties.

In our case study, we use STPM (Figure 5(a)) applied in the context of a web browsing application that frequently uses data which is cached on the disk to display web pages on a handheld device (Anand et al. [2]). DPM decisions are performed as follows: if the HDD is in standby when the browser wishes to fetch a cached object, an *adaptive Cache Manager* estimates the relative time and energy costs of fetching this object from the HDD or the network and selects the device with a smaller cost to retrieve the object. Consequently, for each access, the device with the higher energy cost is allowed to remain inactive longer, thus reducing the overall power (and possibly delay) for the entire application. Each time the browser is forced to use the WNIC to fetch a cached object (since the HDD was inactive), it issues a ghost hint to the HDD. After several such hints, the HDD is proactively woken up on the assumption that more such accesses may occur. More details can be found in [2].

5.2 Experimental Setup

We validate the estimates provided by our model against the following trace-based simulation setup. We extended Krashinsky's ns-2 PSM mode model (Section 2.2) to implement a HDD, cache manager and STPM for both devices. For the WNIC, we used the power and delay values of a Cisco Aironet 350 wireless card [2] and for the HDD we used a 1 GB Hitachi Microdrive and a 5 GB Toshiba drive as examples. Power and delay costs of the HDDs were characterized using synthetic traces within DEMPSEY (**D**isk **E**nergy **M**odeling and **P**erformance **S**imulation **E**nvironment) [29], a set of tools that perform highly detailed stimulus-driven hard-disk simulations for power/performance measurement of HDDs. Using DEMPSEY, we fitted linear regression models that estimate the power and delay of HDDs in terms of the number of blocks read/written. These regression models were then used in ns-2 when estimating the HDD time and energy costs. Network hints required by STPM such as start and end of each network transfer and transfer sizes were obtained from the cumulative density functions (CDFs) of the Berkeley Home IP traces [11] generated by Krashinsky [15]. In the context of the STPM architecture, the various DPM policies that we evaluate include: (i) always use HDD for cached object, WNIC in CAM mode; (ii) always use HDD, WNIC in PSM mode; (iii) always use HDD, WNIC in PSM-ADAPTIVE mode - an adpative algorithm that proactively transitions the WNIC to CAM if there is more than

Table 1. Design Space for different DPM Policies

DPM Policy	# States	CPU Time (sec)	
		Ours	Simulation
Always HDD-CAM	67025	0.6	T.O.
Always HDD-PSM	150615	1.3	T.O.
Always HDD-PSM Adaptive	322953	3.9	T.O.
Cache Manager	605815	7.3	T.O.

T.O.: Time Out after 2 min for each state

one packet for it at the AP; and, (iv) STPM with Cache Manager.

5.3 Modeling STPM

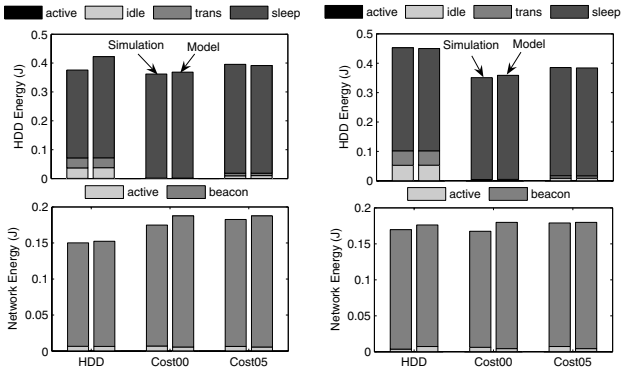
Figure 5(b) illustrates one possible OL-CL-RL model of the STPM architecture. The OL models the workload using two EFSMS: a browser that non-deterministically issues a request for either a cached or a non-cached object and an access point (AP) EFSM similar to the one in the earlier WNIC example. Both EFSMs randomly assume *foreground* and *background* properties to simulate different request characteristics. Transfer sizes for HTTP requests and server responses were abstracted into two categories: LARGE and SMALL using a fixed threshold over the CDFs of these values [11]. The RL represents the WNIC and the HDD using their PSMs. The WNIC PSM models both the CAM and PSM modes while the HDD PSM models the Active, Performance Idle, Low Pwr Idle and StandBy modes [13].

The CL represents STPM as a set of logic formulae that operate on the application hints, device modes, power/delay information and the application state from the OL and RL. The application state specifies if a transfer is a foreground or background transfer and the corresponding transfer size (large or small). For example, $[(\mathsf{APTransfer} = \mathsf{FGND}) \Rightarrow (\mathsf{WNIC} = \mathsf{PSM}) \rightarrow (\mathsf{WNIC} = \mathsf{CAM}))]$ models a foreground transfer hinted by the AP, which causes STPM to transition the WNIC to CAM. The history and counter EFSMs are used to record parameters, like application properties or number of transfers in a given time window, which may help STPM to make a smart decision. The cache manager in the CL is designed to use such information, in addition to the current device power modes, to perform a time and energy cost evaluation of any request to a cached object. The cache manager then selects the device with the lesser cost and provides this "data access decision" to the network and HDD power managers, possibly resulting in a mode transition decision.

5.4 Evaluation of candidate DPM Policies

We ran a web access trace from the Berkeley Home IP traces [11] that accesses 500 unique objects over a 5 Mbps wireless link. The HTTP requests and server responses observed when running this trace in ns-2 were recorded and used to drive the OL EFSMs so that a consistent comparison with the ns-2 simulations could be obtained. Experiments were performed by varying the fraction of cached objects (35% and 50%) and user preference settings ($knob = 0.0$ – performance greedy and $knob = 0.5$ – energy greedy). Different policies were compared using the total energy (the power-delay product) required by each to execute the entire trace.

Figure 6 illustrates the energy consumption comparisons between the conventional simulation and our model for the "ALWAYS HDD" (HDD in Figure 6) and "STPM with Cache Manager" cases. Cost00 and Cost05 in Figure 6 indicate the different user knob values for the cache manager. For the WNIC, we plot only the ACTIVE



(a) 35% Requests Cached (b) 50% Requests Cached

Figure 6. Energy Comparison between the Model and the ns-2 Simulations

and BEACON mode power consumption since the SLEEP mode power dominates due to large user *think times*. Illustrations for the ns-2/DEMPSEY simulations were omitted due to space constraints. The average and peak error by our approach was 3.085%, 12.33% for HDD and 3.954%, 13.76% for the WNIC, respectively. We have obtained similar results in various other experiments involving other DPM policies and more modes for each device.

Table 1 shows the worst case estimation times when performing an exhaustive traversal of the power design space. In this case, no input traces were used, all OL state transitions were performed non-deterministically and the RL state transitions were controlled according to different DPM policies (Table 1-column1). Figure 5(c) illustrates the power envelope of the two-device system due to mode transitions. The CPU time (column 3) in Table 1 is the time per execution step in the model, where all possible state transitions are explored from a given state. Since we are doing this traversal symbolically, the time spent is extraordinarily short when compared with explicit traversal via conventional simulation. All experiments were performed on a Pentium 4 PC with 1 GB RAM running Linux. These results indicate that our high level model can achieve orders of magnitude speedup while accurately tracking the system-wide energy effects of the power management decisions due to different DPM policies. The main advantage of our model from the designer's point-of-view is its ability to describe, evaluate and change the system specification at a very high level of abstraction. If the analysis results are not satisfactory, the designer can quickly modify the DPM policies just by changing the set of rules in the CL.

5.5 Discussion and Ideas for Future Work

A number of lessons were learned during the implementation of our idea. We were constrained in our analysis due to the selection of NuSMV as our input language since an analysis such as ours imposes several constraints that were difficult to handle using NuSMV. First, we found that the calculations used to perform energy and time cost estimation enter the NuSMV model which significantly increases the state space. Second, although state machine-based approaches have been widely used to model concurrent systems, we would also like to explore other forms of representation that may be more amenable to the analysis of DPM architectures. A tool that is more specialized towards such models is more likely to meet these

challenges.

Although we have found our current framework to be sufficient for handling a diverse set of components with a sizeable number of power modes, we intend to study techniques to address state space explosion to allow modeling of more complex systems at a finer timing granularity. We also plan to study different techniques for specification of more meaningful workload models (possibly derived from actual traces) to drive the system model. This is still ongoing work and we believe that there are multiple avenues that can be explored by using such a framework.

6 Related Work

System-level DPM has been the subject of a large body of work for a number of years [3,4,12]. To put our contributions in the proper context, we focus only on formal methods for DPM in this section. For a thorough survey of several ad-hoc techniques, the reader is referred to [4].

Formal techniques for DPM have been used to develop strategies that *theoretically* guarantee bounds on the efficiency of the DPM architecture to achieve power reduction without degrading performance [12]. These can be categorized into stochastic optimum control schemes [5, 18, 20, 22, 25] and competitive analysis-based methods [21, 24]. Stochastic approaches make probabilistic assumptions about usage patterns, idle times etc. to formulate DPM as a stochastic optimization problem which is solved to derive the optimal strategy. However, the resulting policies are not guaranteed to be implementable [4] and the quantitative assessments performed are limited by the assumed probability distributions. Our approach makes no assumptions on usage patterns, workload characteristics and interarrival times, but considers them implicitly using non-deterministic transitions. Competitive analysis based techniques are used to guarantee that the power dissipation due to the derive policies is no more than a certain factor of an oracle policy for a given system. However, these techniques have only been applied to the design of component-level policies [21, 24] and their performance when applied to large systems is unknown.

Our proposed approach takes a middle path between these formal and ad-hoc techniques. We use formal models to describe the system at a high level and then offer a method of pratically investigating system-level tradeoffs. Our work can be closely related to the state-based system-on-chip (SOC) power analysis technique proposed by Bergamaschi and Jiang [7]. They use a PSM-based approach to describe concurrent, communicating cores on a SOC and use a BDD-based symbolic representation and algorithm to exhaustively traverse the design space. We have essentially extended their approach into a framework that is capabale modeling entire systems, including workloads.

7 Conclusions

We presented a formal framework to represent and reason about the system-level tradeoffs of complex systems in the context of DPM policy design. The framework allows designers to efficiently represent both the workload and components of a system using a multi-layered model based on EFSMs. To perform an exhaustive analysis of the impact of DPM decisions on the system-wide energy, we coupled a BDD-based symbolic simulation engine to our model that allows large state spaces to be efficiently traversed. We demonstrated the efficiency of our methodology by using it to model a highly generic and flexible OS-level DPM architecture. Our results indicate that such a formal modeling framework can provide a basis for early analysis and exploration of more aggressive and holistic DPM architectures to tackle the growing problem of limiting system power consumption.

Acknowledgements

We would like to thank Sumeet Sobti and Fengzhou Zhang for providing the DEPMSEY code. We also thank Jason Flinn for providing the STPM source code and traces used in our experiements. This work was supported in part by NSF grants 0121416 and 0219801.

References

[1] M. Anand, E. B. Nightingale, and J. Flinn. Self–tuning Wireless Network Power Management. In *MOBICOM*, pages 176–189, 2003.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the Machine: Interfaces for Better Power Management. In *MOBISYS*, pages 23–35, June 2004.

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

[4] L. Benini, A. Bogliolo, and G. De Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Trans. VLSI*, 8(3):299–315, 2000.

[5] L. Benini et al. Policy Optimization for Dynamic Power Management. *IEEE Trans. CAD*, 18(6):813–833, 1999.

[6] L. Benini, R. Hodgson, and P. Siegel. System-level Power Estimation and Optimization. In *ISLPED*, pages 173–178, August 1998.

[7] R. Bergamaschi and Y. W. Jiang. State-Based Power Analysis for Systems-On-Chip. In *DAC*, June 2003.

[8] B. C. Brock and K. Rajamani. Dynamic Power Management for Embedded Systems. In *IEEE SOCC*, pages 416–419, Sept. 2003.

[9] L. Cai and Y. H. Lu. Joint Power Management of Memory and Disk. In *DATE*, pages 86–91, March 2005.

[10] A. Cimatti et al. NuSMV2: An Open Source Tool for Symbolic Model Checking. In *CAV*, July 2002.

[11] S. D. Gribble. UC Berkeley Home IP HTTP Traces. `http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html`.

[12] R. K. Gupta, S. Irani, and S. K. Shukla. Formal Methods for Dynamic Power Management. In *ICCAD*, pages 874–881, November 2003.

[13] Hitachi Global Storage Technologies. *Hitachi Microdrive Hard Disk Drive Specifications*. January 2003.

[14] IEEE Local and Metropolitan Area Network Standards Committee. *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*. IEEE Std. 802.11-1997. New York, 1997.

[15] A. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. *ACM/Kluwer Jnl. Wireless Networks (WINET)*, 11(1-2):135–148, 2005.

[16] A. S. Krishnakumar and K-T. Cheng. On the computation of the set of reachable states of hybrid models. In *DAC*, pages 615–620, 1994.

[17] D. Li, P. H. Chou, and N. Bagherzadeh. Mode Selection and Mode-Dependency Modeling for Power-Aware Embedded Systems. In *ASP-DAC*, Jan. 2002.

[18] G. Normal, D. Parker, M. Kwaitkowska, S. Shukla, and R. Gupta. Using Probabilistic Model-Checking for Dynamic Power Management. In *Workshop Automated Verfication of Critical Systems*, April 2003.

[19] J. Pouwelse, K. Langendoen, and H. Sips. Application-directed Voltage Scaling. *IEEE Trans. VLSI*, 11(5):812–826, October 2003.

[20] Q. Qiu, and M. Pedram. Dyanmic Power Management Based on continous-time Markov decision processes. In *DAC*, pages 555–561, June 1999.

[21] D. Ramanathan, and R. Gupta. System Level Online Power Management Algorithms. In *DATE*, March 2000.

[22] Z. Ren, B. Krogh, and R. Marculescu. Hierarchical Adaptive Dynamic Power Management. *IEEE Trans. Computers*, 54(4):409–420, April 2005.

[23] K. Shimizu, D. L. Dill, and A. J. Hu. Monitor-Based Formal Specification of PCI. In *FMCAD*, 2000.

[24] S. Shukla, and R. Gupta. A Model Checking Approach to Evaluating System Level Power Management Policies for Embedded Systems. In *HLDVT*, 2001.

[25] T. Simunic. Dynamic Management of Power Consumption. In *Power Aware Computing*. Kluwer, 2002.

[26] T. Simunic, L. Benini, and G. DeMicheli. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. In *DAC*, 1999.

[27] F. Somenzi. CUDD: CU Decision Diagram Package, 2004. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.

[28] H. Touati et al. Implicit state enumeration of finite state machines using BDD. In *ICCAD*, 1990.

[29] J. Zedlewski et al. Modeling Hard-Disk Power Consumption. In *Proc. FAST*, pages 217–230, March, 1990.