Energy-Aware Task Scheduling and Dynamic Voltage Scaling in a Real-Time System

Peng Rong and Massoud Pedram
University of Southern California
Dept. of Electrical Engineering
Los Angeles, CA 90089
e-mail : pedram@ceng.usc.edu

Abstract - This paper presents a solution to the problem of minimizing energy consumption of a computer system performing periodic hard real-time tasks with precedence constraints. In the proposed approach, dynamic power management and voltage scaling techniques are combined to reduce the energy consumption of the CPU and devices. The optimization problem is initially formulated as an integer programming problem. Next, a three-phase heuristic solution, which integrates power management, task scheduling and task voltage assignment, is provided. Experimental results show that the proposed approach outperforms existing methods by an average of 18% in terms of the system-wide energy savings.

I. INTRODUCTION

A dichotomy exists in the design of modern microelectronic systems: they must be low power and high performance, simultaneously. The goal of low-power design for battery-powered electronics is to extend the battery service life while meeting performance requirements. Unless optimizations are applied at different levels, the capabilities of future portable systems will be severely limited by the weight of the batteries required for an acceptable duration of service. Additionally, in fixed power-rich platforms, the packaging cost and power density/reliability issues associated with high power and high performance systems force designers to look for ways to reduce power consumption. Thus, reducing power dissipation is a design goal even for non-portable devices since excessive power dissipation results in increased packaging and cooling costs as well as potential reliability problems. Meanwhile, following Moore's Law, integrated circuit densities and operating speeds have continued to go up in unabated fashion. The result is that chips are becoming larger, faster, and more complex and because of this, consuming increasing amounts of power.

Dynamic power management (DPM) and dynamic voltage scaling (DVS) have both proven to be highly effective techniques for reducing power dissipation in such systems. DPM refers to a selective shut-off of idle system components whereas DVS slows down underutilized resources and decreases their operating voltages.

Much research has been conducted on optimizing power management policies, resulting in both heuristics and stochastic approaches. While the heuristic approaches are easy to implement, they do not provide and power/performance assurances. In contrast, the stochastic approaches guarantee optimality under performance constraints although they are more complex to implement. A number of stochastic models have been reported for DPM. More precisely, to overcome the

limitations of heuristic “time-out”-based power management techniques [1][2], an approach based on discrete-time Markovian decision processes (DTMDP) was proposed in [3]. This approach outperforms the previous heuristic techniques because of its solid theoretical framework for system modeling and policy optimization. However, the discrete-time model requires policy evaluation at periodic time intervals and may thus consume a large number of power dissipation even when no change in the system state has occurred. To surmount this shortcoming, an approach based on continuous-time Markovian decision processes (CTMDP) was proposed in [4]. The policy change under this model is asynchronous and thus more suitable for implementation as part of a real-time operating system environment. Reference [5] also improved on the CTMDP modeling technique by using time-indexed semi-Markovian decision processes (SMDP). An SMDP is a stochastic process where the next state depends on the current state and how long the current state has been active.

In [6], the authors present a hierarchical scheme for adaptive DPM under non-stationary service requests, where the term “hierarchical” refers to the manner by which the authors construct a DPM policy. More precisely, this work assumes that the service providers are fully controllable and have not built-in power management policy. The authors formulate the policy optimization as a problem of seeking an optimal rule that switches policies among a set of pre-computed ones. In contrast, in [7], the authors adopt a hierarchical power management architecture which aims to facilitate power-awareness in an electronic system with multiple components, each equipped with its own local power management structure. The authors thus decompose the power management task into two sub-tasks: component-level and system-level. The component-level policy is pre-specified and fixed although it can be made to behave differently under system commands. The system-level policy optimization is formulated as a concurrent service request flow regulation and application scheduling problem and is solved by the hierarchical power manager.

Previous DVS works can be broadly divided into two categories, one for non real-time operation and the other for real-time operation. The most important step in implementing DVS is prediction of the future workload, which allows one to choose the minimum required voltage/frequency levels while satisfying key constraints on energy and QoS. As proposed in [8] and [9], a simple interval-based scheduling algorithm can be used in non real-time operation. This is because there is no timing constraint. As a result, some performance degradation due to workload mis-prediction is allowed. The defining characteristic of the interval-based scheduling algorithm is that uniform-length intervals are used to monitor the system utilization in the previous intervals and thereby set the voltage level for the next interval by extrapolation. This algorithm is effective for applications with predictable computational workloads such as audio [10] or other digital signal processing intensive applications [11]. Although the interval-based scheduling algorithm is simple and easy to implement, it often predicts the future workload incorrectly when a task’s workload exhibits a large variability. One typical example of such a task is MPEG decoding. In MPEG decoding, because the computational workload varies greatly depending on each frame type, repeated mis-predictions may result in a decrease in the frame rate, which in turn means a lower QoS in MPEG. More recently, a number of researchers have reported DVS algorithms taking into account energy consumption of the system components. In [12] the authors present a DVS heuristic based on the *critical speed* of each task, which is defined as the CPU speed at which the execution of a task consumes the least total system energy. Reference [13] proposes a DVS technique based on a precise energy model considering both the active power and standby component of the system power.

There are also many studies to apply DVS in real-time application scenarios. Multi-task scheduling in the operating system (OS) is the focus of [14] and [15]. More precisely, the scheduling is performed so as to reduce energy consumption while meeting given hard timing constraints. In these coarse-grained DVS approaches, it is assumed that the total number of CPU

cycles needed to complete each task is fixed and known a priori. This is an assumption that is difficult to satisfy in practice. In [16], an intra-task voltage scheduling technique was proposed in which the application code is divided into many segments and the worst-case execution time of each segment (which is obtained from a static timing analysis) is used to determine a suitable voltage for the next segment. In [17] a method based on a software feedback loop was proposed. In this method, a deadline for each time slot is provided. The authors calculate the operating frequency of the processor for the next time slot depending on the slack time generated in the current slot and again the worst-case execution time of the next time slot. In these two fine-grained DVS approaches, it is assumed that the worst-case execution time of each segment of a task is known. This assumption is again difficult to meet for many applications, for example, for MPEG decoding where the worst-case execution time cannot be accurately determined on a per-frame basis. Note that a single worst-case execution time for all video frames (which may be calculated rather easily) will not be useful because it tends to be too pessimistic and therefore will significantly reduce the energy saving potential of a DVS technique that uses it. In [18], an effective DVS algorithm for MPEG decoding is proposed in which the future workload is predicted on a per-frame basis. This is accomplished by using a frame-type-based workload-averaging scheme where the prediction error due to statistical variation in the workload of the frame-dependent part of the decoder is effectively compensated for by using the frame independent part of the decoding time as a "buffer zone." This allows the authors to obtain a significant energy saving without any notable QoS degradation.

Most researches on low-power task scheduling focus only on reducing the CPU power by using DVS techniques. However, in reality, executing a useful task on a computer system requires cooperation between the CPU and many other system components, e.g., memory, disk drives, wireless devices, etc., which can also consume significant amounts of power. These components generally have their own voltage levels and may or may not support DVS, which makes it difficult to apply DVS techniques to the CPU only and achieve total system power savings. In fact, DVS when applied to CPU only may even increase the overall system energy consumption for executing a given set of tasks. At the same time, DPM is known to be an effective approach for reducing the power consumption of the various peripheral components and I/O devices. Thus DVS combined with DPM has the potential to achieve power savings, not possible by either DPM or DVS.

Most related work on low power scheduling for dependent tasks concentrate on DVS techniques. Some authors have considered voltage assignment on distributed embedded systems. The approach proposed in [19] first schedules tasks based on a list-scheduling algorithm by using the reciprocal of the slack time as the task priority, and next tries to evenly distribute the available positive slack time among tasks on each critical path and thereby reduce the operating voltages and save energy. Reference [20] assumes a given task schedule and assignment and proposes an extended list-scheduling algorithm. At each time step, the energy saving of a task is calculated as the difference between the expected energies given the task is scheduled at this step or at the next step. A task with a higher energy saving and lower slack time has a higher priority. The authors of [21] present a two-phase solution. In the first phase, a version of the early-deadline-first scheduling is used to assign a task to a best-fit processor in terms of the task ready time and the processor free time. In the second phase, an ILP optimization problem is formulated and solved in order to determine the voltage level of the processor used to run each task.

Several works on DPM-based task scheduling have also been proposed in the literature. An online scheduling algorithm for independent tasks is presented in [22]. This algorithm attempts to reduce the number of device on/off transitions by greedily extending the pattern for current device usage so as to reduce average power consumption in the near future. Reference [23] proposes an offline branch-and-bound algorithm to search for the energy optimal task scheduling. In [24], the

authors prove that solving energy optimal task scheduling for DPM on multiple devices is an NP hard problem even for a simple case where no timing dependency is considered. References [25] and [26] start with a given timing-fixed task sequence and propose algorithms to determine an energy-minimal state transition sequence for devices while satisfying hard time constraints.

In the literature, several works have been proposed on combining DVS and DPM. Reference [27] present a Markovian decision processes based DPM model which is a uniform modeling framework for both DVS and DPM. In [28], the authors combine DVS with their previously proposed renewal theory based DPM approach. These two stochastic approaches are unable to handle tasks with hard-time constraints or dependency. The problem of combining DVS and DPM for hard real-time tasks is studied in reference [29], where a scheduling algorithm for a single processor with a sleep state is presented which is proved having a competitive ratio of 3. Task dependency is not considered in this work either.

To the best of our knowledge, no research work has been conducted to combine DVS and DPM techniques for hard real-time dependent tasks running on multiple devices. This is specifically the contribution of the present paper. More precisely, this paper addresses the problem of power optimization of a real-time system having heterogeneous components and performing periodic hard real-time tasks. The dependencies between the tasks are described by a directed acyclic graph (DAG), sometimes referred to as a *task graph*. An integer programming based formulation is first provided to exactly state the optimization problem to be addressed. Next, a three-phase algorithm is proposed to solve the power-aware task scheduling and voltage-to-task assignment problems with the objective of minimizing the total system energy consumption. The three steps are power-aware task scheduling, task-level voltage assignment, and task rescheduling and voltage level refinement.

The remainder of the paper is organized as follows: The problem formulation is presented in section 2. The three steps of the proposed algorithm are described in sections 3, 4 and 5, respectively. Experimental results and conclusions are given in sections 6 and 7.

II. PROBLEM FORMULATION

This paper targets a real-time system which has a single CPU and κ system devices (e.g., various I/O devices, main memory.) The CPU is considered to be device number 0 whereas other devices are numbered from1 to κ. The CPU has a discrete number of *performance states* corresponding to different supply voltage levels and clock frequencies and one *sleep* state. All other devices have a *functional* state during which they provide service and a low power *sleep* mode during which they cannot provide any services. [1] Furthermore, a device which is in the performance/functional state can be in one of two sub-states: 1) actively performing services; 2) waiting for service requests to arrive. We will refer to sub-state 1 as the *active* state and sub-state 2 as the *idle* state. We assume that each device k consumes the same amount of power when they are in active or idle mode (denoted by $funcpow_k$), but significantly less power when it is in the sleep mode ($sleepow_k$.)

A set of n non-preemptive dependent tasks periodically run on the system with a time period T_d. The data dependency (precedence) constraints between the tasks are described by a directed acyclic task graph, called a *task graph*, $G(V, E)$, where each node v denotes a task and a directed edge $e(u, v)$ represents a data flow between task u and v and implies that task v can be executed only after task u finishes. Every task has to be performed on the CPU, and may require support (services) from some (or all) of the system devices. It is assumed that during the run time of a task, all devices whose services are required by the task in question will stay in their active modes. The

[1] It is straight-forward to extend the mathematical formulation to handle I/O devices with multiple low-power states (e.g., standby, drowsy, and sleep.)

problem is to solve the optimal task scheduling and task-level voltage assignment with the objective of minimizing the total system energy consumption during period T_d.

Let V_i, i =1, …, m, denote the m operating voltages for the CPU and f_i the clock frequency of the CPU at voltage V_i. We define the workload of task u as the number of CPU cycles without considering memory and IO device access delay. Let $N_{u,i}$ denote the actual number of CPU cycles required to complete task u at operating voltage V_i. We define variable $x(u, i)$ to represent the percentage of the workload of task u which is performed at voltage V_i. Note that there are $m \cdot n$ such variables. The execution time (*duration*) of task u is calculated as

$$dur_u = \sum_{i=1}^{m} \frac{x(u,i) \cdot N_{u,i}}{f_i}, \qquad (2\text{-}1)$$

where

$$\sum_{i=1}^{m} x(u,i) = 1 \cdot \qquad (2\text{-}2)$$

We introduce $n.\kappa$ 0-1 integer variables, $Z_k(u)$, as follows: $Z_k(u) = 1$ exactly if task u requires service from device k. The energy consumption due to execution of task u is equal to

$$ene_u = c_u \sum_{i=1}^{m} x(u,i) \cdot N_{u,i} \cdot V_i^2 + \sum_{k=1}^{K} Z_k(u) \cdot P_k \cdot dur_u \qquad (2\text{-}3)$$

where c_u is the effective switched capacitance per CPU cycle; and P_k is the power consumption of device k in the active mode.

Let $s(u)$ denote the start time of task u. Thus the precedence constraint is expressed as

$$s(u) + dur_u \le s(v) \quad \forall e(u,v) \in E \qquad (2\text{-}4)$$

To formulate the energy consumed by the CPU and devices during idle time, we need to introduce two virtual (dummy) tasks: task 0 of duration zero which is placed at exactly the start of period T_d and task n+1 of duration zero which is placed at the end of period T_d. We define tasks 0 and n+1 so as to require all devices, i.e., $Z_k(0) = 1$ and $Z_k(n+1) = 1, \forall k$. Notice that the interval between task 0 and the first task executed on device k denotes the first idle period. Similarly, the last idle period is defined as the interval between the last task executed on device k and task $n+1$. Also notice that.

We introduce $(n+2)^2 k$ 0-1 integer *scheduling variables* $Y_k(u,v)$ as follows: $Y_k(u,v)$ = 1 exactly if task u is executed on device k immediately before task v is executed on the same device. Since on each device, every task has only one immediate successor, the following constraint on $Y_k(u,v)$ should be respected

$$\sum_{v=1}^{n+1} Y_k(u,v) = \begin{cases} 1, & Z_k(u) = 1 \\ 0, & \text{otherwise} \end{cases}, \quad u = 0,1,...,n \qquad (2\text{-}5)$$

Similarly, every task has only one immediate predecessor; i.e.,

$$\sum_{u=0}^{n} Y_k(u,v) = \begin{cases} 1, & Z_k(v) = 1 \\ 0, & \text{otherwise} \end{cases}, \quad v = 1,2,...,n+1 \qquad (2\text{-}6)$$

There is also a precedence constraint between task v and its immediate successor, both of which are executed on device k, as follows

$$\sum_{u=0}^{n} (s(u) + dur_u) \cdot Y_k(u,v) \le s(v) \quad \forall v \in V, k \in devs_v \qquad (2\text{-}7)$$

With variable $Y_k(u,v)$, we can express the duration of the idle time of device k just before it provides service to task v, $it_{k,v}$, as

$$it_{k,v} = s(v) - \sum_{u=0}^{n} (s(u) + dur_u) \cdot Y_k(u,v) \cdot \qquad (2\text{-}8)$$

Let function $idlene_k(it)$ return the energy consumed by device k during idle time of length it. Note that the device may be placed in a low-power state during its long idle times, as suggested, for instance, in [25][26]. For the illustration purpose, assume that device k has two power states: active and sleep. Let p_a and p_s denote the power consumptions of the device in the active and sleep states, respectively. Let ε_{tr} and τ_{tr} denote the summation of energy overheads and latency overheads associated with the two transitions into and out of the sleep state, respectively. Recall that the breakeven time is equal to $\tau_{BE} = \varepsilon_{tr} / (p_a - p_s)$. Then,

$$idlene_k(it) = \begin{cases} \varepsilon_{tr} + p_s \cdot it, & it \geq \max(\tau_{BE}, \tau_{tr}) \\ p_a \cdot it, & \text{otherwise} \end{cases}$$

Thus, the total energy consumed by the system during time period T_d is calculated as

$$E_{sys} = \sum_{u=1}^{n} ene_u + \sum_{k=0}^{K} \sum_{v=1}^{n} idlene_k(Z_k(v) \cdot [1 - Y_k(0,v)] \cdot it_{k,v}) + \sum_{k=0}^{K} idlene_k(it_{k,n+1} + \sum_{v=1}^{n} Y_k(0,v) \cdot it_{k,v}) \qquad (2\text{-}9)$$

Notice that when executing a periodical task set, for a device, the idle time before the first task starts and the idle time after the last task finishes actually constitute a single idle period. The third term on the right-hand side of equation (2-9) calculates the device energy consumption for such an idle period. The second term on the RHS handles all the other idle times.

The optimization problem is to minimize E_{sys} with respect to constraints (2-1) to (2-8). Note that in this formulation, we ignore the energy and timing overhead associated with the voltage changes because switching of the CPU voltage normally takes between 10-100 microseconds depending on the hardware support for the DVS function. This is negligible compared to the device on/off transition times, which tend to be in the range of a few tenths of a second. The corresponding energy overhead is also small.

This problem is a nonlinear non-convex integer program over variables $s(u)$, $x(u,i)$ and $Y_k(u,v)$; the worst-case computational complexity of exactly solving this problem is expected to be exponential. So we propose a three-step heuristic approach to solve the problem as follows:

Task Ordering: Derive a linear ordering of tasks (i.e., calculate $Y_k(u,v)$ values) by performing an interactive minimum-cost matching on some appropriately constructed graph (cf. section 3.)

Voltage Assignment: Given the task ordering implied by the schedule obtained in step 1, assign voltages and task durations (i.e., calculate $x(u,i)$ values) and exact start times (i.e., calculate $s(u)$ values) to each task so as to meet a target cycle time, T_d (cf. section 4.)

Refinement: Improve the task scheduling and voltage assignment of steps 1 and 2 to increase the energy efficiency of the resulting solutions (cf. section 5.)

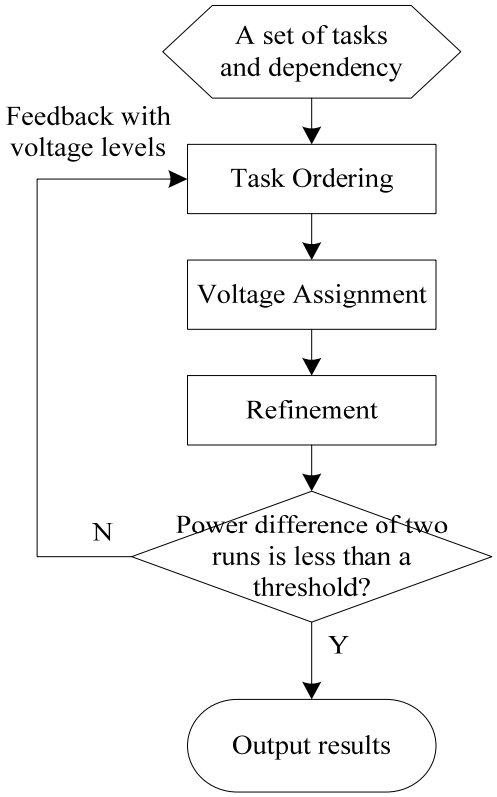


Figure 1. Diagram of the proposed three-step iterative approach.

III. TASK ORDERING

In this step, we assume that the CPU voltage level is set to the maximum possible value and that the task execution times (durations) are calculated on this basis.[2] The goal is to take the task graph with known task execution times and schedule it on the CPU (device 0) so as to minimize the total energy dissipation due to I/O devices (1,…, κ) staying in the idle mode and that caused by transitioning the devices from their high-power functional state to the low-power sleep state. Notice that the summation of energy dissipation in all devices (0,…, κ) when these devices are in active states is fixed and independent of the scheduling. The scheduling only changes the duration of the idle times and the number of on to off transitions for the I/O devices.

Let $tasks_k$ denote the set of tasks running on device k and dev_u denote the set of devices that are needed by task u. A lower bound on the total system energy dissipation, $totene_{LB}$ can be obtained by assuming that there is no energy overhead for the transitions between idle and sleep states of any device

$$totene_{LB} = \sum_{u \in V} \sum_{k \in dev_u} funcpow_k \cdot dur_u + \sum_{k=0}^{\kappa} sleepow_k \cdot (T_d - \sum_{u \in tasks_k} dur_u)$$

Now, the actual total energy includes the energy consumed by various devices when they stay in their idle modes and when they transition in and out of the sleep modes. Let's denote the schedule, Λ, by the start times of all tasks in the given task graph. Based on this information, one

[2] This is a simple heuristic used to assign task durations for this step. Other heuristic assignments are possible. Note, however, that we are only interested in the ordering of tasks after the completion of this step and will in fact calculate the exact task schedule and execution times after voltage assignment.

can linearly order the set of tasks and represent the active times of each device as a set of closed intervals. More precisely, device k will be represented by a segment set, $S_k=\{s_{k,1},\ldots,s_{k,z}\}$ ($z\leq n$) corresponding to the time intervals during which the device is in its active state.

$$t_{nonactive}(s_{k,i},s_{k,i+1}) \equiv \begin{cases} start(s_{k,i+1}) - end(s_{k,i}) & \text{if } i < |S_k| \\ T_d + start(s_{k,1}) - end(s_{k,|S_k|}) & \text{otherwise} \end{cases}$$

$$F_{k,i} \equiv F_k(s_{k,i},s_{k,i+1}) = \begin{cases} 1 & \text{if } t_{nonactive}(s_{k,i},s_{k,i+1}) \geq t_{BE,k} \\ 0 & otherwise \end{cases}$$

$$totene(\Lambda) = totene_{LB} + \sum_{k=1}^{\kappa} \sum_{i=1}^{|S_k|-1} (F_{k,i} \cdot transene_k + (1-F_{k,i}) \cdot funcpow_k \cdot t_{nonactive}(s_{k,i}))$$

Here *start(s)* and *end(s)* denote the start time and end time of segment s while $transene_k$ denotes the total transition energy cost of device k to go from idle mode to the sleep mode *and* to return to the active mode. Next we construct an *augmented task graph* (ATG) $A(V,C)$ from the given task graph $G(V,E)$ by copying $G(V,E)$ and subsequently adding/deleting some edges to/from E. More precisely, the new edge set, C, does not contain any directed edge uv such that there exists another directed path from u to v in C. In addition, C contains undirected edges qr if tasks associated with q and r can be scheduled next to each other in some order. In addition, each node, q, in V (task) has three attributes: task execution time, dur_q, task energy consumption, ene_q, and the list of devices that are required by the task, dev_q. Finally, each directed edge qr in C, has an associated energy cost, $extraene_{qr}$, calculated as follows

$$s_{k,q} \equiv [start(r) - dur_q, start(r)]$$

$$s_{k,succ(r)} \equiv [start(r) + dur_r, T_d]$$

$$F_{k,q,r} \equiv F_k(s_{k,q}, s_{k,succ(r)})$$

$$extraene_{qr} = \sum_{k \in dev_q - dev_r} \sum_{i=1}^{|S_k|-1} (F_{k,q,r} \cdot transene_k + (1-F_{k,q,r}) \cdot funcpow_k \cdot t_{nonactive}(s_{k,i}))$$

Each undirected edge between nodes q and r will have two such energy costs corresponding to directed edges qr and rq. Note however that at most one of the two directed edges may be chosen as part of the scheduling solution.

The basic flow of the proposed scheduling algorithm is to iteratively find the edge with the *least* extra energy value and merge its two end nodes, implying that the corresponding tasks will be scheduled to run in immediate succession. For a directed edge, the ordering is fixed a priori whereas for the undirected edge, the algorithm will choose one of the two possible orderings and fix it. After each merge, the ATG is updated by removing all edges that become invalid and calculating the attributes for the newly generated node. The process continues until a single node is left in the ATG, which corresponds to a total ordering (scheduling) of all the tasks. The process continues until exactly one node remains in the modified ATG (i.e., a complete schedule is obtained.) If at any step of the algorithm, there is a tie between the extra energy costs of two candidate edges qr and uw, then we will choose the edge that would result in the minimum *total extra energy cost*, $extraene_{tot}$, of the resulting graph if the merge was performed. Now, $extraene_{tot}$ is calculated as the summation of the node weights of the resulting graph where the node weight is itself calculated as the average of the extra edge costs of outgoing edges from that node.

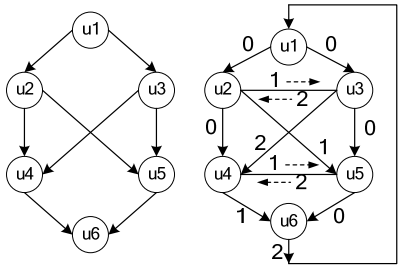
Example 1: Consider a task graph depicted in Figure 2(a). Assume that there are four devices {0,1,2,3} with the following device utilization sets:

$dev(u1) = \{0\}$

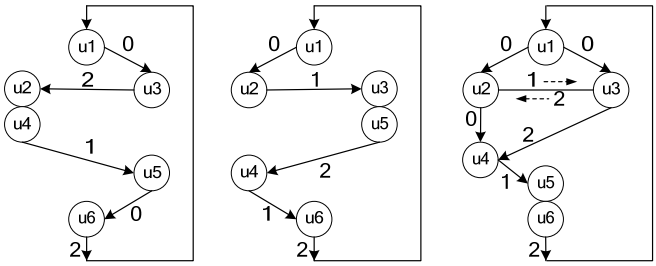
$dev(u2) = dev(u4) = \{0,1\}$.

$dev(u3) = dev(u5) = dev(u6) = \{0,2,3\}$

For the sake of simplicity, we assume that each task has a unit time duration (which is longer than its breakeven time) and that the idle power consumption of all devices is the same. In addition, each device consumes 1 unit of energy for each transition to and from the sleep state. The ATG graph of this task set is given in Figure 2(b). The directed edge $u1u2$ exists in ATG, because there is a precedence constraint between nodes $u1$ and $u2$ and $u2$ can be scheduled immediately after $u1$. The presence of undirected edge $u2u3$ implies that $u2$ and $u3$ can be scheduled next to the other without any ordering constraint. The edge labels denote the energy consumption if the start and end nodes of the edge are scheduled one after the other. For simplicity, assume all node energies are 0.



(a) Initial task graph (b) ATG



(c) ATG after merging a pair of node

Figure 2. An example to illustrate the power-aware task scheduling.

With this ATG, we can start the task scheduling for power management. There are five edges with minimal edge energy equal to 0. That is, we can merge the pair of nodes associated with each of these edges without incurring additional energy cost. In Figure 2(c), three ATGs are presented, each corresponding to the merging of the node pair for one of the edges. Let us consider the left-most ATG which is generated after merging $u2$ and $u4$. Since there is a directed edge from $u2$ to $u4$, $u4$ must be scheduled after $u2$. After the $u2$-$u4$ merge, the edge from $u3$ to $u2$ becomes a directed edge, because originally $u3$ had to be executed before $u4$ which has now been merged with $u2$ into a single node. The directed edge from $u1$ to $u2$ in the initial ATG disappears because after the $u2$-$u4$ merge, $u3$ stands between $u1$ and $u2$ in the precedence chain. The left-most ATG in Figure 2(c) has the minimal E_{ATG} value equal to 5. So the merger of $u2$ and u4 is selected for the first step.

IV. VOLTAGE ASSIGNMENT

Having generated the task schedule, we fix the ordering of tasks, but otherwise, ignore the task execution times and start times, which were heuristically set as explained at the beginning of

section 3, we can easily calculate the $Y_k(u,v)$ values. Thus by substituting the value of $Y_k(u,v)$ into the optimization problem formulated by (2-1) through (2-9), all constraints becomes linear constraints and the only unknowns become $s(u)$ and $x(u,i)$ variables. However, the optimization problem cannot be solved exactly and efficiently, because the objective function remains a non-convex function of idle times, *it.* We thus propose two approaches to get around this non-convexity issue.

The first one simply ignores the energy components introduced by $idlene_k$ in equation (2-9). The optimization problem thus becomes a linear programming problem over continuous variable $x(u,i)$, which can be solved in polynomial time. It is worth pointing out that, strictly speaking, $x(u,i)$ should take discrete values instead of continuous ones, because the number of CPU cycles executed at each operating voltage is an integer. However, when we consider a task executed in hundreds of thousands of CPU cycles, the effect introduced by rounding up to one cycle can be safely ignored.

The second approach introduces new 0-1 integer variables $W_k(v,h)$ to approximate the idle time $it_{k,v}$ as follows: $W_k(v,h)$ =1 exactly if $t_{k,h} \leq it_{k,v} < t_{k,h+1}$. $t_{k,h}$ and $t_{k,h+1}$ take values from a discrete set $\{t_{k,1}, t_{k,2}, \ldots, t_{k,H}\}$. The value of $idlene_k(it)$ is approximated by

$$idlene_k(it_{k,v}) = \sum_{h=1}^{H} idlene_k(t_{k,h}) W_k(v,h), \text{ with } \sum_{h=1}^{H-1} W_k(v,h) = 1.$$

Consequently, constraint (2-8) is thus rewritten as

$$\sum_{h=1}^{H-1} t_{k,h} W_k(v,h) \leq s(v) - \sum_{u=0}^{n} (s(u) + dur_u) \cdot Y_k(u,v) \leq \sum_{h=1}^{H-1} t_{k,h+1} W_k(v,h).$$

Equation (2-9) becomes an integer linear cost function. As a result, the original optimization problem is approximated with a mixed integer linear program. The value of parameter H can be adjusted to trade-off the computational complexity and approximation accuracy.

V. REFINEMENT

We provide a top-level overview of an algorithm that we have developed to improve the results obtained by the first two steps. Starting from the solution obtained from steps 1 and 2, we shift the tasks together to remove redundant positive slack times. Next, we apply a greedy refinement algorithm on this solution to improve the total energy cost while meeting the timing constraint. In particular, we identify the set of critical tasks whose duration has a large impact on the system energy dissipation, e.g., a small change of the duration could enable device transitions to low power states, and change their voltage assignments accordingly. Details are provided next.

Each approach presented in Section 4 tries to obtain a solution by simplifying the formulation of the original optimization problem. Thus the error in the objective function introduced by the simplification method may cause the results to deviate from the optimal solution. In this section, we present a heuristic algorithm to improve the results obtained by the first two steps.

The purpose of this algorithm is to adjust the length of the idle periods to reduce the idle energy consumption. The algorithm has two major steps: 1) slack time redistribution, 2) task voltage adjustment to remove critical idle period.

Slack time redistribution

A slack period is defined as a period in which no task is executed. This step tries to collect available slack times and redistribute them to idle periods where this time can be used to reduce idle energy significantly, e.g. enabling a device to transit into and out of a lower power state. The algorithm of redistributing slack times is presented in Figure 3.

Algorithm for slack time redistribution

Input: A task sequence with $s(u)$, $x(u,i)$ and $W_k(v,h)$ values solved in the step of voltage assignment for each task u.

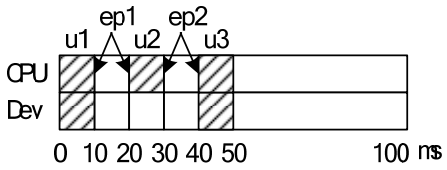
Output: Adjusted values of task star time $s(u)$.

1. *total_asl* = 0.
2. For each task u starting from the leftmost
3. Assuming task v immediately follows task u, the slack immediately after task u is $sl(u) = s(v) - (s(u) + dur_u)$.
4. For each idle duration $it_{k,v'}$ that crosses this slack period
5. Calculate $ir(k,v') = it_{k,v'} - \sum_{h=1}^{H-1} t_{k,h} W_k(v,h)$.
6. The available slack time that can be extracted $asl(u) = \min(sl(u), \min_{k,v'}(ir(k,v')))$.
7. For each task u' after u, $s(u') = s(u') - asl(u)$.
8. *total_asl = total_asl+asl*(*u*).
9. At each epoch separating two tasks or a task and a slack time, we check to see whether or not the total energy consumption can be reduced by inserting a slack time. We select the epoch with the maximal ratio of decrease in energy to increase in slack time. Let *msl* ≤ *total_asl* denote the increase in slack corresponding to the maximal energy ratio.
10. For each task u starting after this epoch, $s(u) = s(u) + msl$.
11. *total_asl = total_asl–msl.*
12. Goto line 9 until *total_asl*=0.
13. Let denote the time corresponds to
14. If $\eta_{w,k} \le D$, then $\boldsymbol{\tau}_{k+1} = \boldsymbol{\tau}_k - (s_k \cdot \nabla\eta_\gamma)/\|\nabla\eta_\gamma\|$. If $\eta_{w,k} > D$, then $i^* = \arg\max_i \{(\partial\eta_\gamma/\partial\tau_{k,i})/(\partial\eta_w/\partial\tau_{k,i})\}$, $\tau_{k+1,i^*} = \tau_{k,i^*} - s_k$.
15. If $\eta_{w,k} \le D \le \eta_{w,k-1}$ or $\eta_{w,k-1} \le D \le \eta_{w,k}$, then set $s_{k+1} = s_k/\alpha$, where $\alpha > 1$ is a constant factor.
16. If both $\eta_{w,k}, \eta_{w,k-1} > D$, then increment a counter *nc* by one. In this case, if *nc* becomes greater than a preset threshold, then set $s_{k+1} = \alpha s_k$ and clear *nc*.
17. If both $\eta_{w,k}, \eta_{w,k-1} \le D$ and $\nabla\eta_{\gamma,k} \cdot \nabla\eta_{\gamma,k-1} < 0$, then set $s_{k+1} = s_k/\alpha$.
18. Go to step 2.

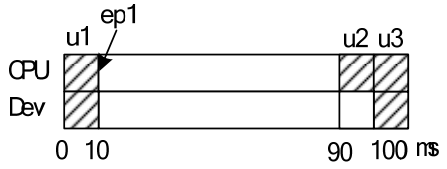
Figure 3: The slack time redistribution algorithm.

***Example 2*:** Consider a task schedule obtained after scheduling and voltage assignment, as shown in Figure 4(a). There are 3 tasks: $u1$, $u2$ and $u3$. The system has a CPU and a device, each having two states: active and sleep. Assume the CPU and the device each consumes 1W in their active states and 0 in their sleep states. Furthermore, it takes 80ms for either one to go to sleep and wake up (e.g., 40 ms to go to sleep and 40 ms to wakeup) and the energy consumption for the transitions is 40mJ. In the initial schedule, neither the device nor the CPU can sleep during each

slack time. During the first step of the refinement procedure, all slack times are collected and the available slack time is calculated to be 80ms. Next, this slack time is allocated to the epoch which has the largest ratio of energy saving per unit of additional slack time. In this example, epoch ep1 and ep2 have the same ratios, so the slack time is arbitrarily allocated to one of them, say ep1. The schedule after refinement is shown in Figure 4(b) where the device can go to sleep while the CPU has to remain active for the whole period. The total energy of the initial schedule is 200mJ whereas it is 160mJ after the refinement. This savings comes about because the device can transit to its sleep state after the refinement.



(a) Initial schedule



(b) Schedule after refinement

Figure 4: Illustrative example for refinement step I

Task voltage adjustment

Third, we adjust the task operating frequency to increase critical idle periods. A critical idle period indicates a long idle period which, however, still cannot accommodate a device state transition. Note that after the first two steps of the flow described in section 4, there may still be some critical idle periods. For a critical idle period, we tentatively increase the operating frequency of the two tasks that limit the length of the idle period, so that the idle period is long enough to enable a device to enter the lower power mode. We select and increase the critical idle period that brings the largest energy saving. This procedure is repeated until no critical idle period can be improved.

***Example 3*:** Let's revisit the previous example. In the schedule obtained after the first two steps of the refinement procedure, i.e., Figure 4(b), the length of the idle period on the CPU is 70ms, which is less than the timing overhead of 80ms for the CPU to enter and leave the sleep state. Now we assume the CPU has a higher operating voltage level, under which the CPU consumes 4W, and the execution time for each task is 5ms. Thus, we can increase the CPU speed to reduce the task execution times, and thereby, create more idle time for the CPU. In the schedule shown in Figure 5, we set tasks *u1* and *u2* to run at the higher CPU speed. Thus the current CPU idle time is increased from 70ms to 80ms, which enables the CPU to transit to the sleep state. The total system energy consumption thus is further reduced to 145mJ.

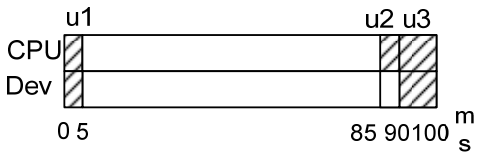


Figure 5: Illustrative example for refinement procedure II

VI. EXPERIMENTAL RESULTS

This experiment is conducted on a system comprising of a single CPU and three other devices. The CPU has three operating voltage/frequency levels: 1V/200MHz, 1.1V/300MHz and 1.3V/400MHz [13]. The CPU and all devices support only one low-power sleep state. The power consumptions in different states, energy and timing overheads of state transitions for both the CPU and the three devices are reported in Table 1.

TABLE 1

Power and transition parameters

Device	Active Power	Sleep Power	Energy Overhead	Timing Overhead
SDRAM	0.3W	~0	~0	~0
HDD	2.1W	0.85W	0.6J	400ms
WLAN	0.7W	0.05W	0.04J	100ms
CPU	1.0W (200MHz)	0.05W	0.3J	400ms

To evaluate the effectiveness of our proposed approach, we generated five task graphs by using software package, TGFF [30], which is a randomized task graph generator widely used in the literature to evaluate the performance of scheduling algorithms. Each task graph consists of 20 to 200 tasks. All tasks require supports from SDRAM. The dependency of tasks on HDD and WLAN were randomly generated and fixed before optimization. The characteristics of different task graphs are given in the following table. For example, for task graph G1, when the CPU has its highest frequency setting, the cpu is used during 61% of the total execution whereas the SDRAM, HDD and WLAN are used for 61%, 29% and 42% of the total time.

TABLE 2

Characteristics of task graphs

Task Graph	No. of Tasks	CPU and device utilization factors at max speed for the CPU CPU SDRAM HDD WLAN
G1	28	0.61 0.61 0.29 0.42
G2	47	0.40 0.40 0.21 0.12
G3	65	0.72 0.72 0.51 0.39
G4	81	0.63 0.63 0.42 0.17
G5	110	0.34 0.34 0.12 0.23

G6	123	0.51 0.51 0.32 0.33
G7	159	0.48 0.48 0.30 0.25
G8	204	0.55 0.55 0.28 0.36

In this experiment, we compare the total system energy consumptions of the following methods:

M1: No DVS, no DPM. The CPU always operates at the highest voltage level and devices are kept active during the whole execution time. This provides the baseline compare against.

M2: DPM without any task scheduling. Tasks are executed on the CPU (which has assumed its highest frequency and voltage setting) in an un-optimized order based on their ID numbers after they become available. A method similar to the approach in [25] is used to determine the state transition sequences of all devices and the CPU.

M3: DPM with task scheduling. This method is similar to M2, except that our proposed power-aware task scheduling algorithm is used to determine the task execution sequence.

M4: Conventional cpu-driven DVS plus DPM. Similar to M2, except that the task operating voltage is assigned to minimize the CPU power consumption. More specifically, the operating voltage setting for each task is obtained by solving the optimization problem defined in section 2 without considering the energy consumption of devices.

M5: Proposed system-aware DVS plus DPM (which have called, Power-aware Scheduling and Voltage Setting or PSVS for short.) Task scheduling and operating voltage settings are determined through the proposed three-phase framework.

TABLE 3
Normalized energy consumption results for different techniques

Task Graph	M2	M3	M4	M5
G1	0.54	0.50	0.58	0.47
G2	0.37	0.35	0.42	0.33
G3	0.67	0.59	0.63	0.53
G4	0.59	0.50	0.58	0.45
G5	0.28	0.26	0.32	0.25
G6	0.45	0.40	0.47	0.37
G7	0.40	0.35	0.42	0.34
G8	0.43	0.37	0.39	0.33

The energy consumptions of different techniques are compared in Table 3. These values have been normalized with respect to the baseline energy consumption of M1, e.g., for G1, M2 results in total system energy consumption which is 54% of the baseline energy consumption. From this table, it is seen that compared to DPM technique without task scheduling, our proposed DPM with task scheduling can reduce energy consumption by an average of 11%. Furthermore, when this method is combined with our proposed voltage assignment technique (resulting in M5 or PSVS), an additional 9% energy saving is achieved.

VII. CONCLUSION

This paper addresses the problem of minimizing energy consumption of a computer system performing periodic hard real-time tasks with precedence constraints. In the proposed approach, dynamic power management and voltage scaling techniques are combined to reduce the energy consumption of the CPU and devices. The optimization problem is first formulated as an integer programming problem. Next, a three-phase solution framework, which integrates power management scheduling and task voltage assignment, is proposed. Experimental results demonstrate efficiency of the proposed approach.

REFERENCES

[1] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. on VLSI Systems*, Vol. 4, pp. 42–55, Mar. 1996.

[2] C-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Int. Conf. Computer-Aided Design*, pp. 28–32, Nov. 1997.

[3] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 18, Issue 6, pp. 813-833, Jun. 1999.

[4] Q. Qiu, Q. Wu, and M. Pedram, "Stochastic Modeling of a Power-Managed System – Construction and Optimization," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 10, No. 10, pp. 1200-1217, Oct. 2001.

[5] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event-driven Power Management," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 20, Issue 21, pp. 840-857, Jul. 2001.

[6] Z. Ren, B. H. Krogh, and R. Marculescu, "Hierarchical Adaptive Dynamic Power Management," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 15, Issue 4, pp. 409-420, Apr. 2005.

[7] P. Rong and M. Pedram, "Hierarchical dynamic power management with application scheduling," *Proc. Symposium on Low Power Electronics and Design*, Aug. 2005, pp. 269-274.

[8] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," Proc. 1st Symp on Operating Systems Design Implementation, 1994, pp. 13-23.

[9] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low power CPU," Proc. 1st ACM Int. Conf. Mobile Computing Networking, 1995, pp.13-25.

[10] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," IEEE Journal of Solid-State Circuit, vol. 35, no.11, Nov. 2000, pp.1571-1580.

[11] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach or energy efficient computing," Proc. International Symposium on Low Power Electronics and Design, 1996, pp.347-352.

[12] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems," *Proc. Symposium on Low Power Electronics and Design*, pp. 78-81, 2001.

[13] K. Choi, W. Lee, R. Soma and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," *Computer Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 1, Jan. 2005, pp.18-28.

[14] F. Yao, A. Demers, and S. Shenker, “A Scheduling Model for Reduced CPU Energy,” IEEE Annual Foundations of Computer Science, 1995, pp.374-382.

[15] T. Ishihara and H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processors,” International Symposium on Low Power Electronics and Design, 1999, pp.197-202.

[16] D. Shin, J. Kim, and S. Lee, “Low-energy intra-task voltage scheduling using static timing analysis,” Proc. Design Automation Conference, 2001, pp.438-443.

[17] S. Lee and T. Sakurai, “Run-time power control scheme using software feedback loop for low-power real-time applications,” Proc. Asia-Pacific Design Automation Conference, 2000, pp. 381-386.

[18] K. Choi, W-C. Cheng, and M. Pedram “Frame-based dynamic voltage and frequency scaling for an MPEG player,” *Journal of Low Power Electronics*, American Scientific Publishers, Vol. 1, No. 1, Apr. 2005, pp. 27-43.

[19] J. Luo and N. Jha, “Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems,” *Proc. Asia and South Pacific Design Automation Conference*, pp. 719-26, 2002.

[20] F. Gruian and K. Kuchchinski, “LEneS: task scheduling for low-energy systems using variable supply voltage processors,” *Proc. Asia and South Pacific Design Automation Conference*, pp. 449-55, 2001.

[21] Y. Zhang, X. Hu, and D.Z. Chen, “Task scheduling and voltage selection for energy minimization,” *Proc. Design Automation Conference*, pp. 183-8, 2002.

[22] Y-H Lu, L. Benini and G. De Micheli, “Low-power task scheduling for multiple devices,” *CODES*, pp. 39-43, 2000.

[23] V. Swaminathan and K. Chakrabarty, “Pruning-based energy-optimal device scheduling for hard real-time systems,” *CODES*, pp.175-80, 2002.

[24] Y-H Lu, L. Benini and G. De Micheli, “Power-aware operating systems for interactive systems,” *IEEE Trans. on VLSI Systems*, vol.10 iss.2, pp. 119-34, 2002.

[25] V. Swaminathan and K. Chakrabarty, “Energy-conscious, deterministic I/O device scheduling in hard real-time systems,” *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol.22 iss.7, pp.847-58, 2003.

[26] J. Liu and P.H. Chou, “Optimizing mode transition sequences in idle intervals for component-level and system-level energy minimization,” *Proc. Int'l Conference on Computer Aided Design*, pp. 21-28, 2004.

[27] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," *Proc. Design Automation Conference,* pp. 555-561, 1999.

[28] T. Simunic, L. Benini, A. Acquaviva, P. Glynn and G. De Micheli, “Dynamic voltage scaling and power management for portable systems,” *Proc. Design Automation Conference*, pp.524-529, 2001.

[29] S. Irani, S. Shukla and R. Gupta, “Algorithms for power savings,” *Proc. Symposium on Discrete algorithms*, pp. 37 – 46, 2003.

[30] http://ziyang.ece.northwestern.edu/tgff.