Proc. Of WTR 2004 – 6th Brazilian Workshop on Real-Time Systems
May 14th , 2004

# An Efficient Energy-Optimal Device-Scheduling Algorithm for Hard Real-Time Systems

S. Chakravarthula[2]
Microsoft, srinivac@microsoft.com

S.S. Iyengar*
Louisiana State University, iyengar@bit.csc.lsu.edu

K. Chakrabarty[3]
Duke University, krish@ee.duke.edu

V. Swaminathan[4]
Duke University, ss@ee.duke.edu

* Corresponding Author

## Abstract

*Dynamic Power Management (DPM) of system components has emerged as a leading research area that aims at minimizing energy consumption in battery-powered electronics such as mobile and embedded systems. An interesting problem under DPM is energy-optimal device scheduling that aims at minimizing device energy consumption for a given task set. In this paper, we present an improved version of Energy-optimal Device Scheduler (EDS), an algorithm originally proposed by Swaminathan and Chakrabarty [34] to solve the problem for hard real-time systems. The improved algorithm cuts down execution time by as much as 99% and memory usage by up to 30% for task sets with a low execution time/period ratio. We also present a heuristic method called Device-Energy Optimizer (DEO) that attempts to find near-optimal solutions and performs remarkably well in comparison to EDS.*

## 1. Introduction

Power management is an important design parameter for computing systems such as laptops, cellular phones and other portable electronics. Most of such systems operate on battery power and minimizing their energy consumption would prolong their operational life. Additionally, in certain situations such as a distributed sensor network over a battlefield, replacing battery packs can turn out to be an expensive affair. Under such circumstances, extending operational life by minimizing energy consumption while guaranteeing task efficiency has tremendous advantages.

The processor and the I/O subsystem are two major components that contribute significantly to a computer system's total energy consumption. Cutting down the energy usage of these subsystems would lead to considerable savings in total power consumption. Several hardware and software techniques have been proposed to achieve this goal.

Most processor-based techniques that have been proposed use two methods for saving energy: 1) powering down the processor when it is idle and 2) scheduling different tasks to run at different speeds by varying a processor's execution voltage. Several popular manufacturers are now shipping processors that have multiple operating voltages ([1], [12], [18], and [31]) thereby allowing the OS to dynamically control the energy consumption of the processor(s). Such a technique of OS-controlled energy management of system components is known as dynamic power management (DPM). CPU-centric DPM has been an active area of research for the past several years and many interesting and efficient approaches have been proposed that aim at minimizing the energy consumption of processors ([6], [10], [11], [13], [14], [17], [19], [20], [23]—[26], [29], [33], [35], and [37]).

Most approaches to I/O-centric DPM are probability, timeout, and statistic based.

***** This project is funded in part by NSF and DARPA IT Grants

Stochastic approaches make use of probability to predict future idle times, whereas timeout-based methods shut down devices after a certain period of idle time. (The industry standard ACPI is a popular implementation of timeout-based DPM.) Statistical techniques, on the other hand, make use of past history of a device to anticipate future requests and then change a device's power state accordingly. Table 1 summarizes prior research in the field of DPM.

An interesting feature that is common to all the above-mentioned approaches to I/O-centric DPM is that they use statistical and probabilistic methods for making device-transition decisions and are therefore not always 100% accurate. Consequently, such methods are unsuitable for hard real-time systems, which are characterized by a high degree of uncertainty in the generation of device requests and stringency in tasks meeting their deadlines.

| **Scientist** | **Approach** | **Method Used/Suggested** |
|---|---|---|
| Greenawalt, 1994 [9] | Statistical | Equation modeling for hard disk power management |
| Hwang and Wu, 1997 [17] | Probabilistic | Exponential-average method in conjunction with prediction-miss correction and pre-wakeup mechanisms |
| Benini et al., 1999 [5] | Finite machines | Finite-state, abstract-system model based on Markov decision processes |
| Simunic et al., 1999 [30] | Finite machines | Suggested modifications to Benini et al.'s method: • switching to a semi-Markov model • using a continuous time model |
| Chung et al., 1999 [7] | Probabilistic and Statistical | Adaptive Learning Tree data structure |
| Lu et al., 2000 [22] | Exact solutions (non-probabilistic, non-statistical) | Re-arrange task executions to prolong device idle periods |
| Swaminthan et al., 2001 [32] | Exact solutions | LEDES algorithm – rearranges task executions (online) |
| Swaminathan and Chakrabarty, 2002 [34] | Exact solutions | EDS algorithm (online) – rearranges task executions (offline) |

**Table 1: Summary of prior I/O-centric DPM research**

Swaminathan and Chakrabarty [34] proposed an offline device-scheduling algorithm named EDS (Energy-optimal Device Scheduler) for hard real-time systems. The EDS algorithm works by trying to keep a device busy for as long as possible. It does this by executing tasks with high device usage overlap one after another thereby minimizing energy consumption of the system as a whole. In this paper, we discuss an improved version of the EDS algorithm that was proposed by Swaminathan and Chakrabarty [34]. The new version runs as much as 200 times faster for certain task sets and uses significantly lesser (up to 30%) memory.

## 2. Preliminaries, Notations, and Assumptions

This section describes the problem domain in detail. We also state the notations and assumptions used in this paper.

The energy-optimal device-scheduling (EODS) problem can be stated as follows: Given a task set $t = \{t_1, t_2, \ldots t_n\}$ and an associated device-usage list $dul = \{d_1, d_2, \ldots d_k\}$ for each task, develop a task schedule that minimizes the energy consumption of all devices while guaranteeing that no task misses its deadline. Table 2 lists task and device characteristics.

The following are a few notations used in this paper:

- *H*: hyperperiod of the task set; defined as the least common multiple of the periods of all tasks.
- *n*: number of tasks
- *l*: number of jobs = $\sum_{i = 1 \text{ to } n} H/p_i$ for a periodic task set.
- *k*: number of devices

We make the following assumptions:

- $min(c_i)$ of all jobs is greater than $max\ (tt_i)$ of all devices
- $pw_i > pt_i > ps_i$
- devices can serve requests only in the working state
- a device state transition can take place at any time instant
- a device schedule can be generated from a given a task schedule by determining the state of each device at the start and completion of each job based on the task's *dul*.

| **Task Characteristics** | **Device Characteristics** |
|---|---|
| • $a_i$ – arrival time<br>• $c_i$ – completion time (worst-case execution time)<br>• $p_i$ – period ($1/p_i$ gives the frequency of task arrival) | • a low-power sleeping state<br>• a high-power working state<br>• transition time from one power state to another $tt_i$<br>• power consumed in working state $pw_i$<br>• power consumed in sleep state $ps_i$<br>• power consumed during transition $pt_i$ |

**Table 2: Task and device characteristics**

## 2.1 Computational Complexity

In this subsection we provide a proof that the energy-optimal device-scheduling problem for hard real-time systems is NP-complete. EODS is a typical example of an optimization problem wherein we are trying to minimize the energy consumed by a set of devices for a given task set. In order to simply the proof of NP-completeness, we recast EODS as a decision problem as follows: given a constant *C* and a task set *T* (with release time and deadline constraints for each task) that uses a device set *D*, is there a feasible schedule for *T* such that *D*'s total energy consumption is less than or equal to *C*? Let us call this recast version of EODS as Decision-EODS. To show that EODS is NP-complete, we need to prove that: 1) Decision-EODS is NP; *and* that 2) Decision-EODS is NP-hard.

**Theorem 1**: *Decision-EODS is NP.*
**Proof:** The proof of this theorem is trivial. Given a task schedule, it is very easy to check in polynomial time whether device-energy consumption is indeed at most *C*.

**Theorem 2**: *Decision-EODS is NP-hard.*

**Proof:** To prove that Decision-EODS is NP-hard, we consider a special case where device usage of all tasks is zero, i.e., *D* is a null set. Decision-EODS then transforms to the following problem: Given a task set *T* (with release time and deadline constraints for each task), is there a feasible schedule that guarantees to meet all release-time and deadline constraints? In other words, we have equated Decision-EODS to the famous *non-preemptive scheduling with release times and deadlines* problem, which has been proven to be NP-complete ([8]). Decision-EODS is therefore NP-hard.

Since we have proven that Decision-EODS belongs to class NP and is NP-hard, we can conclude that EODS is NP-complete.

## 3. Energy-Optimal Device Scheduler (EDS)

A naïve straightforward approach to solving the EODS problem is to compute all feasible task schedules, calculate their respective device energy consumptions, and then choose the one that has the minimal value. A tree-based method could be used to implement such an approach. Given an *l*-job set (a job is an instant of a task), a schedule tree consisting of nodes and directed edges could be constructed, wherein each node, identified by a {*id*, *scheduled-time*} tupple, would be a schedulable instant of a released job. By "schedulable instant" we mean a time instant that guarantees a job's completion before its deadline. Directed edges exist between parent and child nodes. A parent node *a* can have a child node *b* if *b* can be scheduled after the completion of *a*. The root of the schedule tree is a dummy node identified by {0,0}. A path of depth *l* from the root node to a leaf node gives a complete feasible schedule.

| ***id*** | $a_i$ | $c_i$ | $d_i$ | ***dul*** |
|---|---|---|---|---|
| $j_1$ | 0 | 1 | 2 | {$d_1$,$d_2$} |
| $j_2$ | 0 | 1 | 3 | {$d_2$,$d_3$} |
| $j_3$ | 2 | 1 | 4 | {$d_1$,$d_2$} |
| $j_4$ | 3 | 1 | 6 | {$d_2$,$d_3$} |
| $j_5$ | 4 | 1 | 6 | {$d_1$,$d_2$} |

**Table 3: Job characteristics**

| ***id*** | $pw_i$ (J/s) | $ps_i$ (J/s) | $pt_i$ (J/s) | $tt_i$ (J/s) |
|---|---|---|---|---|
| $d_1$ | 2.30 | 1.00 | 1.50 | 0.60 |
| $d_2$ | 0.30 | 0.10 | 0.20 | 0.50 |
| $d_3$ | 0.63 | 0.25 | 0.40 | 0.50 |

**Table 4: Device characteristics**

We now describe the generation of a schedule tree in more detail using the job set given in Table 3 (device characteristics are given in Table 4). First, at *time* = 0, we find all released jobs, which are $j_1$ and $j_2$ in our example. We then find all schedulable instants for $j_1$ and $j_2$. $j_1$ can be scheduled at *time* = 0 and *time* = 1 without missing its deadline and $j_2$ can be scheduled at *time* = 0, *time* = 1, and *time* = 2. We therefore create the nodes (1,0), (1,1), (2,0), (2,1), and (2,2) and then draw edges from the root to these nodes. What we are doing is, in essence, converting each job into a set of nodes representing all schedulable instants of that particular job. We proceed to the next level and for each node at that level we generate child nodes in a similar fashion. For example, the node (1,0) tells us that $j_1$ completes at *time* = 1. We find all released unscheduled jobs up to *time* = 1. We see that the only unscheduled released job at *time* = 1 is $j_2$. We compute all schedulable instants of $j_2$, which are {(2,1), (2,2)}, and then draw edges from (1,0) to these nodes. We expand all the other nodes at this level {(1,1), (2,0), (2,1), and (2,2)} in a similar fashion.

Sometimes, no job might have arrived by the time the current job completes. Under such circumstances, we advance current time to the nearest job arrival time. At other times, a node can fail to generate vertices for a released unscheduled job because of deadline problems. Consequently, that particular node can be safely pruned because it will fail to grow into a complete schedule. We call such a method of pruning as *temporal* pruning (pruning based on deadlines). For example, the node (2,1) informs us that $j_2$ completes at *time* = 2 (since $c_{j2}$ = 1), therefore, the earliest time instant any job can be scheduled next is *time* = 2. Also, the jobs released but not scheduled by *time* = 2 are $j_1$ and $j_3$. However, it can be observed that $j_1$ cannot meet its deadline if it were scheduled at *time* = 2 (since $c_{j1}$ = 1 and $d_{j1}$ = 2). We now know that node (2,1) will definitely fail to grow into a complete feasible schedule. It can therefore be safely pruned.

We continue expanding the schedule tree until all nodes have been expanded and we reach the *l*th level. If we fail to reach level *l*, we can conclude that no feasible schedule exists for the given task set. Figure 1 shows the complete schedule tree for the job set given in Table 3. Once we have the set of feasible schedules, we calculate the device-energy consumption in each schedule and then choose the schedule with the minimal value.

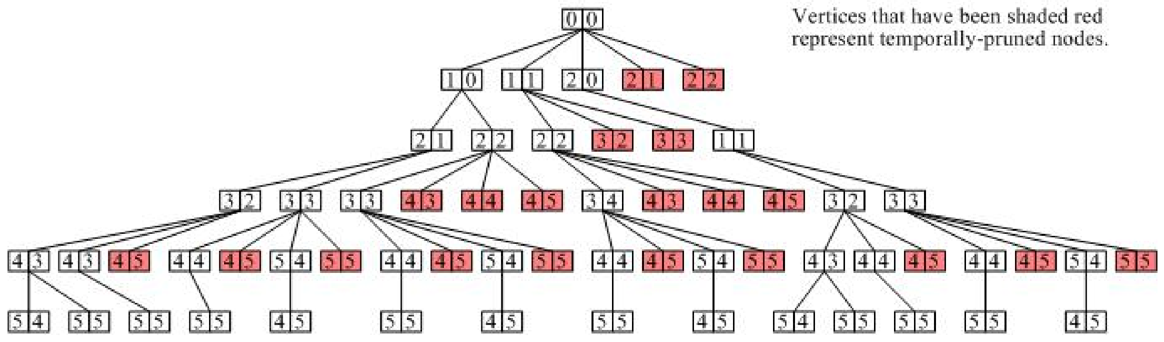

Vertices that have been shaded red represent temporally-pruned nodes.

**Figure 1: Schedule tree for job set given in Table 2.**

It can be observed that a schedule tree grows "factorially" with *l*. We have already proved that EODS is NP-complete and therefore no algorithm exists as of today to solve it in polynomial time. However, [34] identified some approaches that lead to a significant reduction in the size of the schedule tree thereby making larger data sets solvable. They proposed an algorithm called EDS (Energy-optimal Device Scheduler) that implemented their tree-size-reduction techniques with remarkable results. The basic idea behind the EDS algorithm is to prune certain "superfluous" branches of the schedule tree based on energy computation.

Branches are labeled "superfluous" when it can be guaranteed that the optimal solution does not exist along them. Such superfluous branches are identified and removed using a pruning method known as *energy-based* pruning. The second technique they proposed was to treat each node at level 1 as a root node and thereafter proceed with the schedule tree generation for each such root node separately. This, in effect, is like generating sub-problems and solving them separately. The latter technique is directed towards reducing the memory usage of the schedule tree thereby providing scope for solving larger data sets that have higher memory requirement. The interested reader is referred to [34] for details on how *energy-based* pruning is implemented.

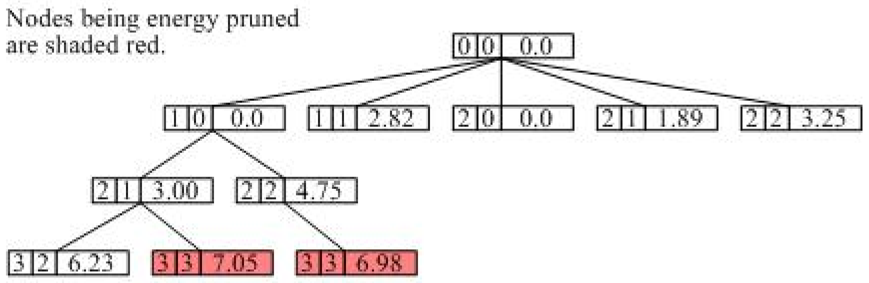


**Figure 2: Partial schedule tree generated using EDS for the job set given in Table 2 (device characteristics are given in Table 3).**

The denser the tree gets the more the number of branches that get pruned as a result of the above techniques thereby yielding remarkable reductions in the algorithm's running time and memory usage in practice. Once the tree is fully developed, a path of depth *l* from the root to a leaf node with the least energy value is the most energy-optimal task schedule (there can be more than one energy-optimal task schedules). Figure 3 gives the complete schedule tree for the job set given in Table 3 (device characteristics are given in Table 4). The most energy-optimal task schedule(s) are indicated using shaded vertices.

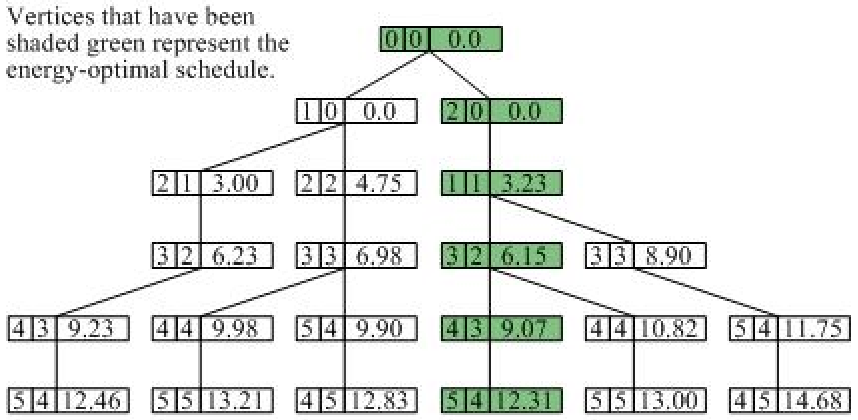


**Figure 3: Complete schedule tree generated using EDS for job set given in Table 2 (device characteristics are given in Table 3).**

## 4. An Improved Version of EDS

In this section, we propose improvements to EDS that make it even more efficient. Our observations led us to two avenues that could potentially lead to improvements in the algorithm. They are: 1) reducing the size of the tree by introducing new methods of pruning, and 2) improving the search time in lines 14 through 21 in the algorithm (Figure 4). The improvements we suggest yield remarkable reductions in running time and memory usage.

The first improvement we propose aims at reducing the number of nodes generated in the tree by adopting a new pruning technique. The pruning method we suggest is called *look-ahead temporal* pruning. In this method of pruning, we avoid generating nodes that will lead to dead ends deeper in the schedule tree. The basic idea behind *look-ahead temporal* pruning is explained next.

Under certain circumstances, even though a job $j_x$ can be scheduled at a specific time instant $s_x$ (with an energy consumption of $e_x$), it may render other released jobs "unschedulable" because of deadline problems and thus lead to a dead end in the task schedule. We would therefore save time and memory if the node ($j_x$,$s_x$,$e_x$) were not generated in the first place. Assume that we are currently expanding a node ($j_c$, $s_c$, $e_c$). We can implement *look-ahead temporal* pruning by adopting the following 3-step procedure for each job $j_{next}$ that can be scheduled next:

1. define a set *UR* of all unscheduled jobs
2. determine the smallest $d_j$-$c_j$ value (= *sd; secondary deadline*) in the set *UR*
3. schedule $j_{next}$ up to $min(d_{next}-c_{next}, sd-c_{next})$

Following the above procedure guarantees that a feasible schedule will be generated (if one exists) while simultaneously implementing *look-ahead temporal* pruning. What we are essentially doing by following the above procedure is determining a secondary deadline for the current job by using the deadlines of all unscheduled jobs. We then use this secondary deadline to determine the maximum schedulable instant for the current job. Doing so will guarantee that the current job will not make a job released later "unschedulable".

We illustrate *look-ahead temporal* pruning using the example job set given in Table 3. Figure 5 shows a partial schedule tree for the example job set. Energy values are not shown because they do not play a role in *look-ahead temporal* pruning. At this point in the development of the schedule tree, we are at level 0 expanding the node {0,0} by processing $j_2$ ($j_1$ and $j_2$ are the jobs that have arrived by time instant 0 and we have already processed $j_1$). The possible nodes that can be generated for $j_2$ at this juncture are {2,0}, {2,1}, and {2,2}. However, if $j_2$ were to be scheduled at time instant 1, we would not be able to schedule $j_1$ later because $d_{j1}$ = 2 and $c_{j1}$ = 1. We should therefore avoid scheduling $j_2$ at time instant 1 and refrain from creating the node {2,1}. A similar argument holds for the node {2,2}. We can determine the maximum schedulable instant for $j_2$ by following the above-mentioned 3-step procedure. For this example, $j_{next} = j_2$, $d_{next} = 3$, $c_{next} = 1$, $UR = \{j_1, j_3, j_4, j_5\}$ and $sd = 1$. As per the above-mentioned metric, the maximum schedulable time instant for job $j_2$ would then be $min(d_{next}-c_{next} = 2, sd-c_{next} = 0)$, which is 0. We can now safely avoid generating the "superfluous" nodes {2,1} and {2,2}.

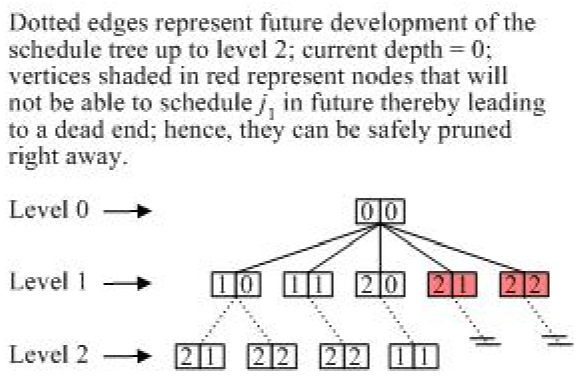

Dotted edges represent future development of the schedule tree up to level 2; current depth = 0; vertices shaded in red represent nodes that will not be able to schedule $j_1$ in future thereby leading to a dead end; hence, they can be safely pruned right away.

**Figure 5: A schedule tree illustrating *look-ahead* temporal pruning for the job set given in Table 2.**

The way the older version of EDS works is that it first creates such “superfluous” nodes, stores them, and deletes them much later only when it has moved on to the next level and has started processing them at that level. The drawbacks with such an approach are: 1) valuable processor time is consumed for creation of the “superfluous” nodes; 2) time spent on the search section of the algorithm increases; and 3) memory usage of the algorithm is higher. The new *look-ahead temporal* pruning approach on the other hand identifies such superfluous nodes well in advance and avoids creating them thus saving time and memory. Section 7 shows experimental proof of its effectiveness in reducing the total number of nodes generated in a schedule tree.

The second improvement we put forward reduces the time spent on the search section of the algorithm (lines 14 through 21 in Figure 4). The way the search section of the older version of EDS works is it first generates all nodes at a particular level, stores them in list, and then does a pair-wise comparison of nodes to eliminate some of them using *energy-based* pruning. This involves an $O(s^2)$ worst-case-time search, where $s$ is the number of nodes at a level. For larger data sets, $s$ can become prohibitively large as we go deeper in the schedule tree thereby exploding the running time of the algorithm in practice. We recommend using a hash-based search to facilitate *energy-based* pruning that brings down the search time to $O(s)$. A hash table can be used to store and search nodes. As and when a node is generated, we avoid duplication by using the hash table to determine if such a node already exists (a node having same *energy-based* pruning criteria as this node, *i.e.*, representing the same job scheduled at the same time instant and having identical ordering of previously scheduled jobs = partial schedule). Consequently, at any point of time, at each level we keep only one node with the least device energy consumption value representing a particular job scheduled at a particular time instant having a particular partial schedule. This reduces the size of the hash table significantly and improves the search time. Our experiments, discussed in Section 7, show that the hash-based search method dramatically improves running time thereby providing scope for solving larger data sets. Such large data sets were previously unsolvable in a reasonable amount of time using EDS. The Improved EDS algorithm is given in Figure 6.

```
Procedure ImprovedEDS(J, l)
J : Job set
l: Number of jobs
vertexQueue: Queue of unexpanded vertices
ht: Hash table of vertices generated in the schedule tree
readyList: List of jobs ready to be scheduled
time: time counter
depth: tree depth
sd: secondary deadline for a job (discussed in section 4)

1.  Set time = 0, depth = 0;
2.  Enqueue vertex (0,0,0) into vertexQueue;
3.  while vertexQueue is not empty {
4.    Dequeue vertex v = {$j_i$,$s_i$,$e_i$} from vertexQueue
5.    Set time = $s_i$+$c_i$;
6.    Update readyList with all jobs released up to time;
7.    for each job j in readyList {
8.      if j has not been previously scheduled {
9.        Generate set UR of all unscheduled jobs excluding j
10.       Set sd = smallest $d_i$-$c_i$ value in the set UR
11.       for t = max(time, $a_j$) to min($d_j$-$c_j$, sd-$c_j$) {
12.         Generate a vertex $v_{new}$ = (j,t,e)
13.         if a v' = $v_{new}$ already exists in ht {
14.           if partial schedule($v_{new}$) = partial schedule(v') {
15.             if  energy value of $v_{new}$ > energy value of v'
16.               Prune $v_{new}$;
17.             else {
18.               Prune v';
19.               Delete v' from ht;
20.               Delete v' from vertexQueue;
21.               Insert $v_{new}$  into ht;
22.               Enqueue $v_{new}$  into vertexQueue;
23.             }
24.           }
25.           else {
26.             Insert $v_{new}$  into ht;
27.             Enqueue $v_{new}$  into vertexQueue;
28.           }
29.         }
30.         else {
31.           Insert $v_{new}$  into ht;
32.           Enqueue $v_{new}$  into vertexQueue;
33.         }
34.       }
35.     }
36. }
37. Clear readyList;
38. Increment depth;
39. If depth = l
40. Terminate;
41. }
```

**Figure 6: The Improved EDS algorithm.**

## 5. A Heuristic Solution

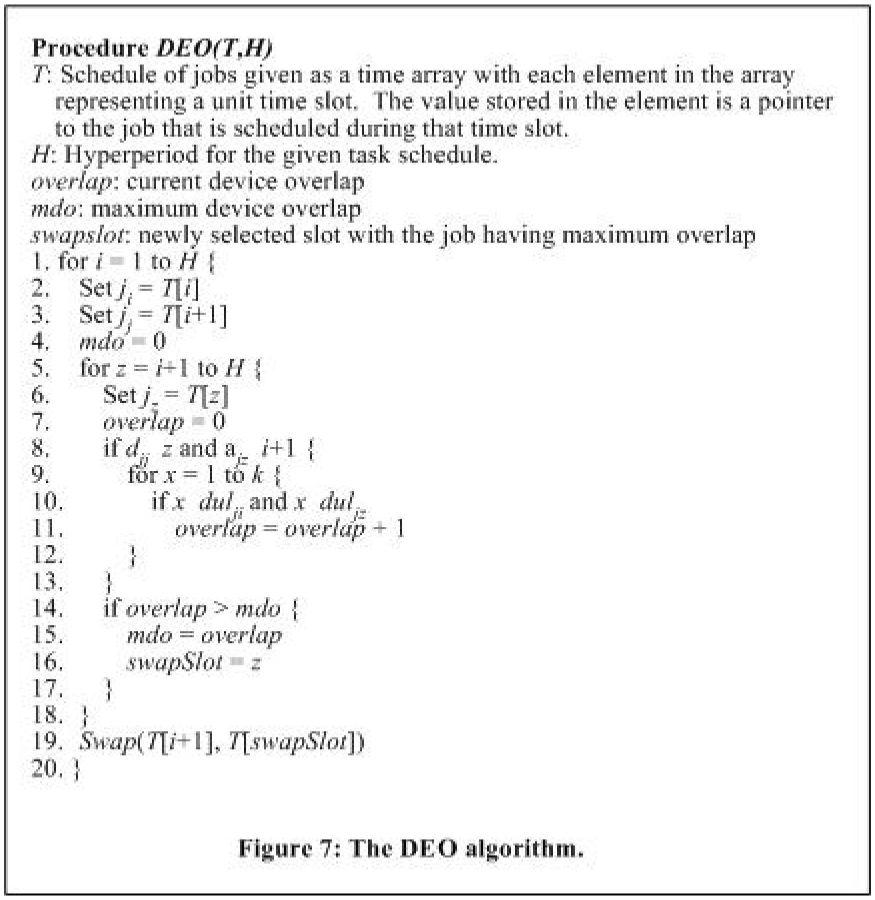
Given that EDS is factorial in time complexity, the improvements suggested in Section 5 might not appear to be that effective. Theoretically, this assumption is quite true since any polynomial improvement to a super-polynomial algorithm is not good enough to bring down the running time considerably. Consequently, finding near-optimal solutions that have polynomial-time complexity would appear to be more logical. In this section, we present a heuristic approach for solving the EODS problem in polynomial-time. The heuristic we propose is called the Device-Energy Optimizer (DEO).

DEO takes a feasible task schedule as input and tries to rearrange task executions such that device energy consumption is minimized. A feasible schedule for a given job set can be easily generated using one of several algorithms such as Earliest Deadline First (EDF). The task schedule given as input to the DEO algorithm is in the form of a time array $T$ with each element of the array representing a time slot. Each element stores a pointer to the job that executes during that time slot. The DEO algorithm works as follows. At the completion of a time slot, DEO scans the rest of the time array to find a 'swappable' slot that has the closest device-usage list to the current job. We illustrate the idea of 'swappable' using the following example. Assume that we just completed executing job $j_i$ during the time slot $T[i]$ and that the job in the next immediate time slot $T[i+1]$ is $j_j$. We now find a job (in a slot $T[z]$) in the range $T[i+2 \ldots H-1]$ that has arrived by $i+1$ *and* has maximum device overlap with $j_i$. We then swap $j_z$ with $j_j$. However, we also have to make sure that $z$ is less than $j_j$'s deadline. The DEO algorithm is given in Figure 7. After the algorithm finishes, the time array T gives us a task schedule with significantly lower device energy consumption. The reason for reduced energy consumption of this task schedule is that devices remain in a certain power state for an extended period of time instead of constantly switching between on and off states, which causes consumption spikes. DEO reduces energy consumption by trying to schedule together jobs that have higher device usage overlap

The running time of DEO is $O(kH^2)$, which is polynomial in time complexity. As discussed earlier, EDS has super-polynomial time complexity and even with all the improvements listed in Section 5, it still takes prohibitively large amounts of time even for data sets with just over 20 jobs. We attempted to solve data sets using both improved EDS and DEO and the results are listed in Table 4. The table compares the performance of improved EDS and DEO and makes it evident how excellent a choice DEO is for solving larger data sets with a little tradeoff in energy consumption.

## 6. Experimental Results

We evaluated the improved EDS algorithm by using task sets (listed in Table 3) with varying hyperperiods (and consequently, varying number of jobs). Device characteristics are listed in Table 4. Table 5 shows the running time and memory usage of the improved EDS algorithm for these data sets. Running time of the algorithms was measured using the ‘time’ command in LINUX. The number of nodes generated was used as a metric for memory usage. Table 5 also lists the running time and memory usage of the original EDS algorithm for the same data sets. A PC with a Pentium3-800MHz processor and 256MB of RAM was used for conducting the experiments.



**Procedure *DEO(T,H)***

*T*: Schedule of jobs given as a time array with each element in the array representing a unit time slot. The value stored in the element is a pointer to the job that is scheduled during that time slot.

*H*: Hyperperiod for the given task schedule.

*overlap*: current device overlap

*mdo*: maximum device overlap

*swapslot*: newly selected slot with the job having maximum overlap

```
1. for $i$ = 1 to $H$ {
2.    Set $j_i = T[i]$
3.    Set $j_j = T[i+1]$
4.    $mdo$ = 0
5.    for $z = i+1$ to $H$ {
6.       Set $j_z = T[z]$
7.       $overlap$ = 0
8.       if $d_{ji}$  $z$ and $a_{jz}$  $i+1$ {
9.          for $x$ = 1 to $k$ {
10.            if $x$  $dul_{ji}$ and $x$  $dul_{jz}$
11.               $overlap = overlap + 1$
12.         }
13.      }
14.      if $overlap > mdo$ {
15.         $mdo = overlap$
16.         $swapSlot = z$
17.      }
18.   }
19.   $Swap(T[i+1], T[swapSlot])$
20. }
```

**Figure 7: The DEO algorithm.**

| Task Set | EDS | | Improved EDS | |
|---|---|---|---|---|
| | Time | Memory Usage (Number of Nodes) | Time | Memory Usage (Number of Nodes) |
| $H$ = 20; $l$ = 9 | < 1s | 75 | < 1s | 42 |
| $H$ = 30; $l$ = 11 | < 1s | 247 | < 1s | 163 |
| $H$ = 35; $l$ = 12 | < 1s | 439 | < 1s | 300 |
| $H$ = 40; $l$ = 13 | < 1s | 799 | < 1s | 590 |
| $H$ = 45; $l$ = 14 | < 1s | 1193 | < 1s | 902 |
| $H$ = 55; $l$ = 16 | 1.03s | 2717 | < 1s | 1909 |
| $H$ = 60; $l$ = 17 | 3.91s | 4045 | < 1s | 2939 |
| $H$ = 65; $l$ = 18 | 17.23s | 6035 | < 1s | 4472 |
| $H$ = 70; $l$ = 19 | 48.43s | 8155 | < 1s | 6205 |
| $H$ = 80; $l$ = 21 | 5m42.71s | 15822 | 4.31s | 11065 |
| $H$ = 85; $l$ = 22 | 14m18.95s | 21043 | 10.05s | 15099 |
| $H$ = 90; $l$ = 23 | 38m53.44s | 28381 | 25.35s | 20541 |
| $H$ = 95; $l$ = 24 | 90m11.04s | 36047 | 1m01.22s | 26668 |
| $H$ = 105; $l$ = 26 | 575m30.36s | 62525 | 5m06.52s | 42889 |
| $H$ = 110; $l$ = 27 | 1479m06.84s | 77889 | 9m36.74s | 54484 |
| $H$ = 115; $l$ = 28 | 4384m25.74s | 98721 | 19m09.83s | 69381 |

**Table 5: EDS versus Improved EDS.**

It is evident from Table 5 that the Improved EDS algorithm cuts down running time by more than 99% and memory usage by up to 34% when compared to the original version. Such improvements are a direct result of implementing *look-ahead temporal* pruning, which is well complemented by the hashing-based search system we suggested. Data sets that earlier took more than a day (for example, the last 2 data sets in Table 5) to be solved can now be run to completion in less than 20 minutes. It should be noted that both the old and the new EDS algorithms are actually super-polynomial in complexity and therefore, as a consequence, no significant theoretical improvements are evident. However, our experimental results prove beyond doubt that the improvement in the practical running time is quite remarkable. These results also suggest that practical improvements to optimal algorithms can be significant even if theoretical improvements are not apparent. Table 5 also suggests that running time of the improved EDS algorithm does not explode as much as that of the old EDS algorithm when data set size increases. The other aspect of the improved EDS algorithm is that it cuts down memory usage by more than 25% for most of the data sets.

Table 5 also suggests that *look-ahead temporal* pruning is most effective in cutting down running time and memory usages for data sets that have tasks with low $c_i/p_i$ ratio, i.e., tasks having greater number of schedulable instants. Such tasks actually generate greater number of nodes in the original EDS algorithm. Most of these nodes run into dead ends deeper in the schedule tree because of deadline problems with other tasks (as discussed in Section 5). *Look-ahead temporal* pruning removes these “superfluous” nodes quite early in the development of the schedule tree thereby cutting down the running time and memory usage. It would seem logical to assume that hard real-time systems would be quite conservative in that the window of scheduling instants they would provide for each task would be quite wide, *i.e.*, for most tasks $c_i$ would be much less than $p_i$ (that is, they would be characterized by a low $c_i/p_i$ ratio). Therefore, if this assumption were true, *look-ahead* temporal pruning would result in tremendous improvements in running time and memory usage in practice for hard real-time systems.

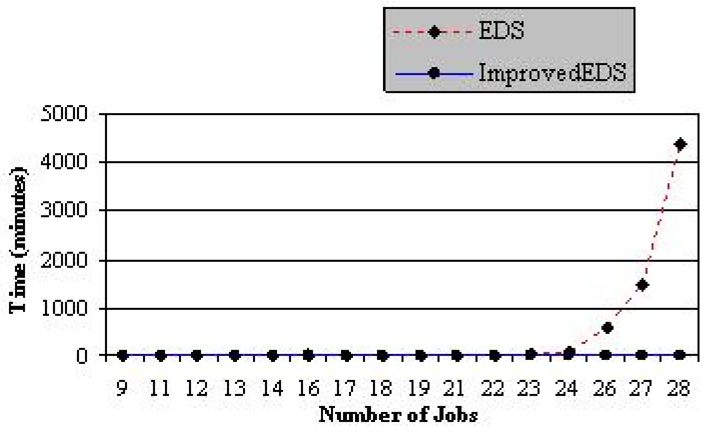


**Figure 8: Running Time Comparison: EDS vs. Improved EDS**

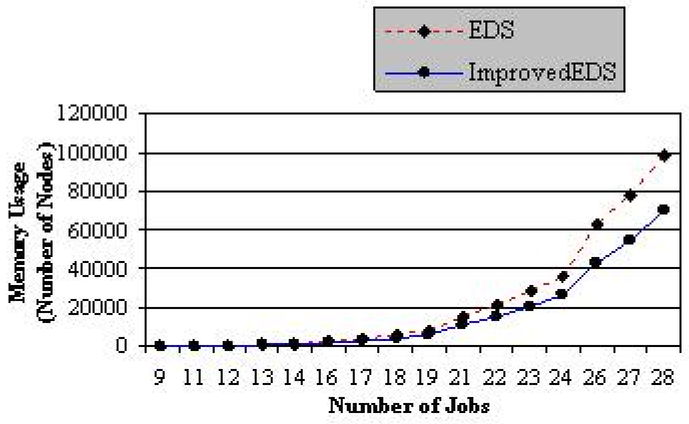


**Figure 9: Memory Usage Comparison: EDS vs. Improved EDS**

Figure 8 shows the rate of growth of running time for EDS and Improved EDS. It can be observed that when number of jobs is beyond 26, EDS explodes in running time. On the other hand, Improved EDS' running time shows minimal increase. Its rate of growth is much less than that of EDS. In fact, Improved EDS takes less than 1/200th the time EDS takes for solving a 28-job set. Figure 9 depicts the rate of growth of memory usage for the two algorithms. Similar to running time, EDS' memory usage increases dramatically when the number of jobs touches 26. Improved EDS on the other hand has significantly lower memory usage for the same job set. It can also be observed that the memory usage of Improved EDS for solving a 28-job set is approximately 30% less than that of EDS. These properties of Improved EDS – reduced rate of growth in running time and memory usage – suggest that the algorithm can be used to solve larger data sets with relative ease when compared to EDS. We highlight these properties as salient features of Improved EDS.

We now discuss how our heuristic solution performs in comparison to EDS. Table 6 lists the results of comparisons between DEO and improved EDS. It is evident that DEO has the ability to solve very large data sets in relatively negligible amounts of time. It can also be seen that task schedules generated using DEO have only a maximum of 5% increase in device energy consumption when compared with those generated using EDS. This 5% increase is an acceptable tradeoff given the savings of up to 99.99% in running time. Figure 10 gives a visual comparison of energy solutions from EDS and DEO. It can be concluded that DEO is a first-rate alternative for EDS when near-optimal solutions to EODS are tolerable.

## 7. Conclusions

Energy conservation in embedded and portable systems is an active area of research currently. Most of such systems depend on battery power for entirety of their life. Bringing down the power consumption of system components would therefore result in an extension of operational life of such battery-powered systems. The I/O subsystem in such systems is as much a powerful

| **Task Set** | **EDS** | | **DEO** | |
|---|---|---|---|---|
| | **Time** | **Energy (Joules)** | **Time** | **Energy (Joules)** |
| *H* = 20; *l* = 9 | < 1s | 44.12 | < 1s | 45.25 |
| *H* = 30; *l* = 11 | < 1s | 60.92 | < 1s | 62.72 |
| *H* = 35; *l* = 12 | < 1s | 69.85 | < 1s | 72.42 |
| *H* = 40; *l* = 13 | < 1s | 78.17 | < 1s | 80.68 |
| *H* = 45; *l* = 14 | < 1s | 87.13 | < 1s | 90.38 |
| *H* = 55; *l* = 16 | 1.03s | 104.33 | < 1s | 106.88 |
| *H* = 60; *l* = 17 | 3.91s | 112.73 | < 1s | 115.13 |
| *H* = 65; *l* = 18 | 17.23s | 121.53 | < 1s | 123.38 |
| *H* = 70; *l* = 19 | 48.43s | 129.93 | < 1s | 131.6 |
| *H* = 80; *l* = 21 | 5m42.71s | 147.13 | < 1s | 148.12 |
| *H* = 85; *l* = 22 | 14m18.95s | 156.00 | < 1s | 156.37 |
| *H* = 90; *l* = 23 | 38m53.44s | 164.33 | < 1s | 164.62 |
| *H* = 95; *l* = 24 | 90m11.04s | 170.45 | < 1s | 172.87 |
| *H* = 105; *l* = 26 | 575m30.36s | 186.23 | < 1s | 189.37 |
| *H* = 110; *l* = 27 | 1479m06.84s | 194.22 | < 1s | 197.47 |
| *H* = 115; *l* = 28 | 4384m25.74s | 200.12 | < 1s | 206.35 |

**Table 6: Improved EDS versus DEO.**

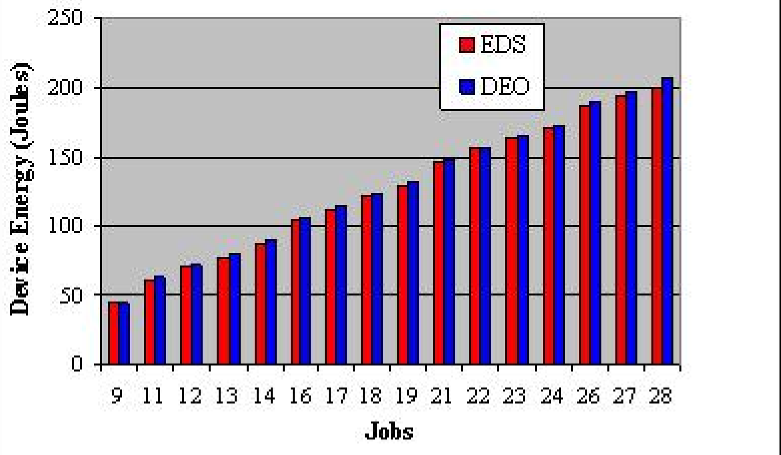


**Figure 10: Device-Energy Consumption: EDS vs. DEO**

candidate as the processor for cutting down energy usage. The I/O subsystem consists of peripheral devices such as display, storage, and communication units. Dynamically (through an OS) changing the power states of such units (i.e., scheduling devices) can minimize device energy consumption.

Energy-optimal device scheduling for hard real-time systems (EODS) has been proven to be NP-complete. The EDS algorithm proposed by [34] was the first attempt to come up with an efficient algorithm to solve the problem. In this paper, we have presented an improved version of EDS that introduces a new method of pruning known as *look-ahead temporal* pruning. We also suggest a change in the search method used by the original EDS algorithm. Our experiments show that these changes bring about a remarkable reduction in running time and memory usage. We also presented a new heuristic algorithm called DEO for finding near-optimal solutions to EODS. Our experimental results show that DEO is an excellent choice for finding fast solutions to EODS problem with very little tradeoff in device energy consumption.

## 9. References

1. AMD. *Athlon 4 Processor Data Reference Sheet #24319*. Advanced Micro Devices, Inc., 2001.
2. Benini, L. and Micheli, G. De. System-level power optimization: techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, pp. 115-192, 2000.
3. Benini, L., Bogliolo, A., and Micheli, G. De. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, pp. 299-316, 2000.
4. Benini, L., Bogliolo, A., Cavalucci, S., and Ricco, B. 1998. Monitoring system activity for OS-directed dynamic power management. *Low Power Electronics and Design*, pp. 185-190, 1998.
5. Benini, L., Bogliolo, A., Paleologo, G., and Micheli, G. Policy optimization for dynamic power optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 813-833, 1999.
6. Brown, J., Chen, D., Greenwood, G., Hu, X., and Taylor, R. Scheduling for power reduction in a real-time system. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 84-87, 1997.
7. Chung E.-Y., Benini, L., and Micheli, G. Dynamic power management using adaptive learning tree. *International Conference on Computer-Aided Design*, pp. 274-279, 1999.
8. Garey, M. and Johnson, D. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman. San Fransisco. 1979.
9. Greenawalt, P. Modeling power management of hard disks. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 62-66, 1994.
10. Gruian F. and Kuchcinski K. LEneS: Task scheduling for low-energy systems using variable supply voltage processors. *http://www.cs.lth.se/Research/ESD/doc/aspdac01.pdf*
11. Gruian, F. Hard real-time scheduling for low-energy using stochastic data and DVS processors. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 46-51, 2001.
12. Hitachi, Inc. *SH1: SH7032/SH7034 Product Brief.*
13. Hong, I., Qu, G., Potkonjak, M., and Srivastava, M. Synthesis techniques for low-power hard real-time systems on variable voltage processors. *Real-Time Systems*

*Symposium*, pp. 178-187, 1998.

14. Hsu, C.-H., Kremer, U., and Hsiao, M. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 275-278, 2001.
15. http://developer.intel.com/technology/iapc/acpi/
16. Hwang, C.-H. and Wu, A. A predictive system shutdown method for energy saving of event-driven computation. *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, pp. 226-241, 2000.
17. Hwang, C.-H. and Wu, A. A predictive system shutdown method for energy saving of event-driven computation. *International Conference on Computer-Aided Design*, pp. 28-32, 1997.
18. IBM Documentation at *http://www-3.ibm.com/chips/products/powerpc/chips/*
19. Ishihara, T. and Yasuura, H. Voltage scheduling problem for dynamically variable voltage processor. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 197-202, 1998.
20. Lee, Y.-H., Doh, Y., Krishna, C. EDF scheduling using two-mode voltage-clock-scaling for hard-real-time systems. *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 221-228, 2001.
21. Lorch, J. and Smith, A. Software strategies for portable computer energy management. 1998. *IEEE Personal Communications*, vol. 5(3), pp. 60-73, 1998.
22. Lu, Y.-H., Benini, L., and Micheli, G. Low-power task scheduling for multiple devices. *International Workshop on Hardware/Software Codesign*, pp. 39-43, 2001.
23. Manzak, A. and Chakrabarti, C. Variable voltage task scheduling for minimizing energy. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 279-282, 2001.
24. Nakamoto Y., Tsujino Y., and Tokura N. Real-time task scheduling algorithms for maximum utilization of secondary batteries in portable devices. *Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 347-354, 2000.
25. Okuma, T., Yasuura, H., and Ishihara, T. Software energy reduction techniques for variable-voltage processors. *IEEE Design and Test of Computers*, vol. 18(2), pp. 31-41, 2001.
26. Pering, T., Bird T., and Broderson, R. The simulation and evaluation of dynamic voltage scheduling algorithms. *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 76-81, 1998.
27. Puterman, M. Finite Markov decision process. *New York: Wiley*. 1994.
28. Quan, G. and Hu, X. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. *Design Automation Conference*, pp. 828-833, 2001.
29. Shin, D., Kim J., and Lee, S. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers*, vol. 18(2), pp. 20-30, 2001.

30. Simunic, T., Benini, L., and Micheli, G. Event-driven power management of portable systems. *Proceedings of the 12th International Symposium on System Synthesis*, pp. 18 –23, 1999.

31. Suziki, K., Mita, S., Fijita, T., Yamane, F., Sano, F., Chiba, A., Watanabe, Y., Matsuda, K., Maeda, T., and Kuroda, T. A 300 MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 587-590, 1997.

32. Swaminathan, V., Chakrabarty, K., and Iyengar, S. Dynamic I/O power management for hard real-time systems. *Proceedings of the International Symposium on Hardware/Software Co-Design*, pp. 237-243, 2001.

33. Swaminathan, V. and Chakrabarty, K. Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems. *Proceedings of the ASP-DAC*, pp. 251-254, 2001.

34. Swaminathan, V. and Chakrabarty, K. Pruning-based energy-optimal device scheduling for hard real-time systems. *Proceedings of the International Symposium on Hardware/Software Codesign*, pp. 175-180, 2002.

35. Yao, F., Demers, A., and Shanker S. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of Computer Science*, pp. 374-382, 1995.

36. Zhang, Y., Hu, X., and Chen, D. Task scheduling and voltage selection for energy minimization. *Design Automation Conference*, pp. 183-188, 2002.