An Integrated Heuristic Approach to Power-Aware Real-Time Scheduling

Pedro Mejía-Alvarez[1], Eugene Levner[2], and Daniel Mossé[3]

[1] CINVESTAV-IPN, Sección de Computación, Av. IPN. 2508, México DF.
pmejia@cs.cinvestav.mx
[2] Holon Academic Institute of Technology, Department of Computer Science,
52 Golomb St, Holon 58102, Israel.
levner@hait.ac.il
[3] Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260.
mosse@cs.pitt.edu

Abstract. In this paper we propose a novel scheduling framework for a dynamic real-time environment that experiences power consumption constraints. This framework is capable of dynamically adjusting the voltage/speed of the system, such that no task in the system misses its deadline and the total energy savings of the system are maximized.
Each task in the system consumes a certain amount of energy, which depends on a speed chosen for execution. The process of selecting speeds for execution while maximizing the energy savings of the system requires the exploration of a large number of combinations, which is too time consuming to be computed on-line. Thus, we propose an integrated heuristic methodology which executes an optimization procedure and an approximate greedy algorithm in a low computation time. This scheme allows the scheduler to handle power-aware real-time tasks with low cost while maximizing the use of the available resources and without jeopardizing the temporal constraints of the system. Simulation results show that our heuristic methodology achieves a performance with near-optimal results.

1 Introduction

Power management is increasingly becoming a design factor in portable and hand-held computing/communication systems. Energy minimization is critically important for devices such as laptop computers, PCS telephones, PDA's and other mobile and embedded computing systems simply because it leads to extended battery lifetime.

The problem of reducing and managing energy consumption has been addressed in the last decade with a multi-dimensional effort by the introduction of engineering components and devices that consume less power, low power techniques involving VLSI/IC designs, algorithm and compiler transformations, and by the design of computer architectures and software with power as a primary source of performance. Recently, hardware and software manufacturers have introduced standards such as the ACPI (Advanced Configuration and Power Interface) [8] for energy management of laptops, desktops and servers that allow several modes of operation, turning off some parts of the computer (e.g., the disk) after a preset period of inactivity.

Energy management is also achieved by *variable voltage scheduling* (VVS), which involves dynamically adjusting the voltage and frequency (hence, the CPU speed). By reducing the frequency at which a component operates, a specific operation will consume less energy but may take longer to complete. Although reducing the frequency alone will reduce the average energy used by a processor over that period of time, it may not always deliver a reduction in energy consumption overall, because the power consumption is linearly dependent on the increased time and quadratically dependent on the increased/decreased voltage.

In the context of dynamic voltage scaled processors, VVS in real-time systems is a problem that assigns appropriate clock speeds to a set of periodic tasks, and adjust the voltage accordingly such that no task misses its predefined deadline while the total energy savings in the system is maximized.

The aim in this work is to study the problem of maximizing energy savings during the scheduling of dynamic real-time tasks in a single processor environment. In a dynamic environment, we must compute a solution for our power optimization problem at every task arrival (and departure). The identification of feasible options that maximize our optimality criteria (expressed as the total energy savings of the system) requires the exploration of a large combinatorial space of solutions. This optimization problem is stated in this paper as a linear (0/1) multiple-choice knapsack optimization problem [16].

In order to cope with the highly computation costs of the dynamic real-time environment, we have developed a low-cost power-aware scheduling paradigm. Our Power-Optimized Real-Time Scheduling Server (PORTS) consists of four stages: (a) an *acceptance test* for deciding if and when dynamically arriving tasks can be accepted in the system, (b) a *reduction procedure* which transforms the original multiple-choice knapsack optimization problem into a standard knapsack problem, (c) a *greedy heuristic algorithms* used to solve the transformed optimization problem, and (d) a *restoration algorithm* which restores the solution of the original problem from the transformed problem. The *optimization procedure* developed (b,c and d above) are novel mathematical formulations which provide a near-optimal solution for the problem of selecting speeds of execution of all tasks in the system. The solution developed satisfies the condition of maximizing the energy savings of the system while guaranteeing the deadlines of all tasks in the system. The performance of the PORTS Server and its heuristic algorithms will be compared with the performance of several known algorithms.

The remainder of this paper is organized as follows. In Section 2 related models and previous work are reviewed. In Section 3, the system and energy models used in this paper are defined. In Section 4, the power-optimized scheduling is formulated as an optimization problem. In Section 5, the Power-Optimized Real-Time Scheduling Sever (PORTS) is described and in Section 6 we describe a methodology for handling power-aware real-time tasks. In Section 7, simulation results are presented to show the performance of the PORTS Server. Finally, Section 8 presents concluding remarks.

2 Related Work on Variable Voltage Scheduling

Broadly speaking, there are two methods to reduce power consumption of processors through OS-directed energy management techniques. The first is to bring a processor into a power-down mode, where only certain parts of the computer system such as the clock generation and the timer circuits are kept running when the processor is in idle state. Most power-down modes have a trade-off between the amount of power savings and the latency overhead incurred during mode change. For an application that cannot tolerate latency, as those in real-time systems, the applicability of power-down modes is limited. The second method is to dynamically change the speed of a processor by varying the clock frequency along with the supply voltage. Power Reduction via variable voltage can be classified as *static* and *dynamic* techniques. Static techniques, such as static scheduling, compilation for low power [17] and synthesis of systems-on-a-chip [7], are applied at design time. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving dynamically arriving real-time tasks, light workloads or the system is idle.

Static (or off-line) scheduling methods to reduce power consumption in real-time systems were proposed in [24, 10, 5]. These approaches address task sets with a single period or aperiodic tasks. Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of off-line periodic requests are proposed in [6]. Non-preemptive power-aware scheduling is investigated in [5]. Recent work on VVS includes the exploitation of idle intervals in the context of the Rate Monotonic and Earliest Deadline First (EDF) scheduling frameworks [19, 11, 2, 15]. Most of the above research work on VVS assumes that all tasks have identical power functions. Using an alternate assumption, efficient power-aware scheduling solutions are provided where each real-time tasks have different power consumption characteristics [1, 4].

Although systems which are able to operate on an almost *continuous* voltage spectrum are rapidly becoming a reality thanks to advances in power-supply electronics [3], it is a fact nowadays that most of the microprocessors that support dynamic voltage scaling use a few *discrete* voltage levels. Some examples of processors that support discrete voltage scaling are: (a) the Crusoe processor [23] which is able to dynamically adjust clock frequency from 200 to 700 MHz and from 1.1 V to 1.6 V, in 33 MHz steps; (b) the ARM7D processor [22] which can run at 33MHz and 5V as well as at 20MHz and 3.3V; and (c) the Intel StrongARM SA1100 processor, which supports 11 clock speeds: 59-221 MHz in 14.7 MHz Steps [9].

3 System and Energy Models

We consider a set $\mathbf{T} = \{T_1, \ldots, T_n\}$ of n periodic preemptive real-time tasks running on one processor. Tasks are independent (i.e., do not share resources) and have no precedence constraints. Each task T_i arrives in the system at time a_i. The Earliest Deadline First (EDF) [13] scheduling policy will be considered.

The *life-time* of each task T_i consists of a fixed number of instances r_i, that is, after the execution of r_i instances, the task leaves the system. The period of T_i is denoted by P_i, which is equal to the relative deadline of the task.

Examples of event-driven real-time systems exhibiting this behavior include: (1) Internet video conferencing and multimedia systems, where media streams are generated aperiodically; each stream contains a fixed number of periodic instances which are transmitted over the network, and (2) digital signal processing, where each task processes source data that often arrives in a *bursty* fashion.

Given a CPU speed determined by a voltage/frequency pair, the worst-case workload is represented by the traditional worst-case execution time (WCET) value. Note that, however, for VVS framework where the actual execution time is dependent on the CPU speed, the worst-case number of required CPU cycles is a more appropriate measure of the workload. We denote by C_i the number of processor cycles required by T_i in the worst-case. Under a constant speed SP_i (given in cycles per second), the execution time of the task is $t_i = \frac{C_i}{SP_i}$. A schedule of periodic tasks is *feasible* if each task T_i is assigned at least C_i CPU cycles before its deadline at every instance. The *utilization* of the system denotes the amount of processor load in percentage that a task is demanding for execution. $U_i = \frac{t_i}{P_i}$ (or $\frac{C_i}{SP_i P_i}$) denotes the utilization of task T_i. According to EDF, a set of tasks are feasible (no tasks misses its deadline) if the utilization of the system is less or equal than the total capacity of the system, $\sum U_i \leq c$. For EDF, $c = 1$; that is, the achievable capacity is 100%.

We assume that at the arrival of any task, the CPU speed can be changed at *discrete* levels between a minimum speed SP_{min} (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed SP_{max}. SP_{ij} denotes the speed of execution of an instance of task T_i when executes at speed j, and U_{ij} denotes the utilization of task T_i executing at speed j. The power consumption of the task T_i is denoted by $g_i(SP)$, assumed to be a strictly increasing *convex* function [3], specifically a polynomial of at least second degree. If the task T_i occupies the processor during the time interval $[t_1, t_2]$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(SP(t))dt$. The total energy consumed in the system from $t = 0$ up to $t = t_2$ is therefore $E(0, t_2)$. We assume that the speed remains the same during the execution of a single instance. Finally, a schedule is *energy-optimal* if it is feasible and the total energy consumption for the entire execution of the system is minimal.

While applying voltage-clock scaling under EDF scheduling, we make the following additional assumptions: (1) *The time overhead associated with voltage switching is negligible.* According to [23] the time overhead associated with voltage switching in the Transmeta Crusoe microprocessor is less than 20 microseconds per step. The worst-case scenario of a full swing from 1.1 V to 1.6 V takes 280 microseconds, and (2) *Different tasks have different power consumptions.* This assumption is based in the real-life fact that the power dissipation is dependent on the nature of the running software of each task in the system. This assumption is clearly justified taking into consideration the following examples: some tasks will use more of the memory system (in addition to the cache), some tasks will use the floating point unit more than others, some will ship the tasks to specialized processors (e.g., DSPs, micro-controllers, or FPGAs).

4 Formulation of the Problem

In a real-time system with energy constraints, the scheduler should be able to guarantee the timing constraints of all tasks in the system and to select the speed of execution of each task such that the energy consumption of the system is minimized, or equivalently, that the energy savings of the system is maximized.

Therefore, the problem can be formulated as follows. Each time a new task T_i arrives or leaves the system, the problem is to determine the speed of execution for each task in the system such that no task misses its deadline and the energy savings of the system is maximized. Note that a solution to this problem must be computed each time a new task arrives or leaves the system, therefore we can not allow a solution with high computation time.

4.1 The Optimization Problem

For each task T_i in the system we define a set of speeds of execution which will be called class N_i. Each level of speed $j \in N_i$ has a Energy Saving computed by

$$S_{ij} = (E_{i1} - E_{ij}) \tag{1}$$

where E_{i1} is the energy consumed by task T_i executing at its maximum speed and E_{ij} denotes the energy consumption of T_i executing at speed j.

Furthermore, each task running at speed SP_{ij}, will have utilization $U_{ij} = \frac{C_i}{SP_{ij} \cdot P_i}$. Note that the size of class N_i is n_i and the total number of items is $m = \sum_{i=1}^{n} n_i$. It is assumed that the items $j \in N_i$ for all tasks are arranged in non-decreasing order, so that S_{i1} and U_{i1} are the items with the smallest values in N_i. Each task T_i in the system accrues an accumulated energy savings S_i^k upon executing a number of instances during the interval of time between arrivals a_k and a_{k+1}. S^k denotes the amount of energy savings accrued by all the tasks in the system during $a_{k+1} - a_k$.

$$S^k = \sum_{i=1}^{n} S_i^k \tag{2}$$

The aim of this *optimization problem* is to find an speed level $j \in N_i$ for each task T_i, such that the sum of energy savings for all tasks is maximized without having the utilization sum to exceed the capacity of the system c. That is,

$$\begin{aligned}
&\textbf{maximize} \quad Z_0 = \textstyle\sum_{i=1}^{n} \sum_{j \in N_i} \; S_{ij} \;\; x_{ij} \\
&\textbf{subject to} \quad \textstyle\sum_{i=1}^{n} \sum_{j \in N_i} U_{ij} \;\; x_{ij} \le c \\
&\textstyle\sum_{j \in N_i} \;\; x_{ij} = 1, \;\; i = 1, ..., n
\end{aligned}$$

$$x_{ij} = \begin{cases} 1 \text{ if speed } j \in N_i \text{ for task } T_i \text{ is chosen} \\ 0 \text{ otherwise} \end{cases}$$

We call this problem, Problem P_0.

By achieving the optimality criteria, whenever a new task arrives or departs from the system, we intend to maximize the accumulated energy savings S^k for each arrival and therefore to maximize the accumulated energy savings obtained after scheduling the entire set of tasks for the complete duration of the schedule.

We have formulated the power saving problem as a Multiple-Choice Knapsack Problem (MCKP) with 0-1 variables [16]. According to the real-life requirements of dynamic power-aware real-time systems, any instance of the *medium-size* MCKP containing 10 to 80 tasks with 5 to 40 different speed levels is to be solved within a few milliseconds. However, the MCKP is known to be NP-hard [16] which implies that it is very unlikely to design a so fast (polynomial-time) exact method for its solution. From a practical point of view it means that some of the available exact methods for power-aware scheduling that solve our optimization problem, such as dynamic programming [16], Lagrange multipliers [1], mixed-integer linear programming [21] and enumeration schemes [6], do not satisfy the above realistic requirements for solving the problem.

5 PORTS: Power-Optimized Real-Time Scheduling Server

The Power-Optimized Real-time Scheduling Server PORTS, is an extension of the Earliest Deadline First scheduling algorithm (EDF [13]). The PORTS Server is capable of handling dynamic real-time tasks with power constraints, such that the energy savings of the system is maximized and the deadlines of the tasks are always guaranteed. In order to meet our optimality criteria, when new tasks arrive in the system, the PORTS Server adjusts the load of the system by controlling the speed of execution of the tasks.

The PORTS Server is activated whenever a new task arrives in the system. The PORTS Server first executes a Feasibility Test (FT) to decide whether or not the new task can be accepted for execution in the system. If the new task is accepted, an *optimization procedure* is executed to calculate the speeds of execution of all tasks in the system.

This optimization procedure consists of three parts:

1. A *reduction algorithm*, which converts the original MCKP to a standard KP.
2. An *approximation algorithm* (e.g. Enhanced Greedy Algorithm) capable of finding an approximate solution to the reduced KP, and
3. A *restoration algorithm*, which re-constructs the solution of the MCKP from the KP.

The solution provided by the *optimization procedure* is such that no task in the system misses its deadline and the speeds of execution chosen for all tasks, maximizes the energy savings of the system. After the *optimization procedure* is executed, the Total Bandwidth Server [14] is used to compute the start time of the new task. Finally, with the start time of the new task computed and the solution provided by the optimization procedure (the set of speeds for execution), the PORTS Server will schedule the new task in the system.

The PORTS Server is also activated when a task leaves the system, in which case, the Feasibility Test is not executed.

6 Handling Power-Aware Real-Time Tasks

The proposed method consists of five basic parts, or stages, as illustrated in Figure 1, and described in detail in the following subsections.

6.1 Activating the PORTS Server and Feasibility Test

The two conditions for activating the PORTS Server and their procedures are:

1. Task Arrival. When a new task T_j arrives in the system, the feasibility test is executed. The task is rejected when running all tasks (including T_j) at the maximum speed (minimum utilization) the system is not feasible. Otherwise, the new task is accepted:

Feasibility Test (FT):

$$FT = \begin{cases} T_j \text{ is accepted } if\ U_{min} = \sum_{i=1}^{n} \ U_{i1} \ \leq 100\ \% \\ T_j \text{ is rejected } \text{ otherwise} \end{cases}$$

After a new task has been accepted in the system, the next problem is to choose the speed of execution of each task in the system. This problem is related to our optimization problem because by choosing a speed for the execution of task T_i we will obtain its corresponding energy savings achieved. Obviously, energy savings are minimum when all tasks execute at their maximal speeds. Therefore, our goal is to choose the speed for execution of each task such that our optimization criteria is met.

2. Task Departure. The PORTS Server is also activated when a task leaves the system. In this case, the optimization procedure is executed to satisfy the optimality criteria for the new set of tasks in the system. In this case, the Feasibility Test is clearly not needed.

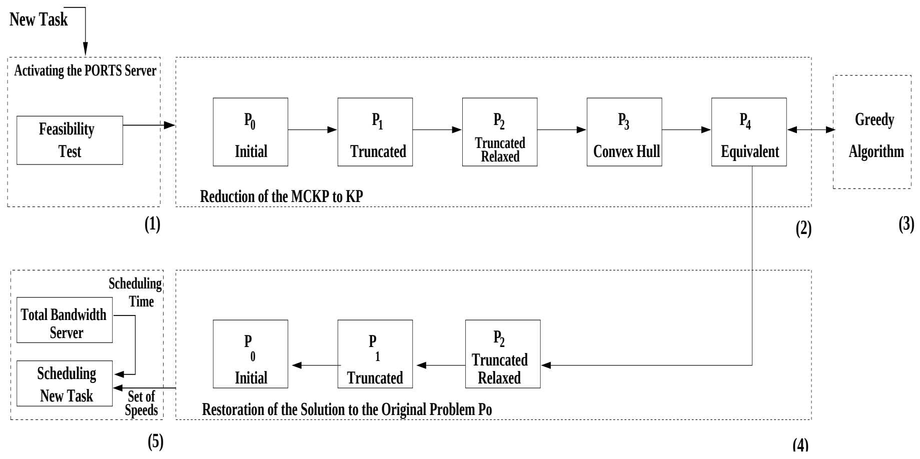

Fig. 1. Methodology for Handling Power-Aware Real-Time Tasks

6.2 Reduction Scheme from MCKP to the Classical KP

Our approximation algorithm is based on the reduction of the MCKP to the equivalent KP using the convex hull concept [16]. In order to reduce the MCKP, denoted by P_0, the following auxiliary problems will be used:

P_1: The Truncated MCK Problem

Problem P_1 is constructed from P_0, by extracting the lightest item from each class and assuming that all these items are inserted into the knapsack. The sum of the lightest items from each class is denoted by $S_0 = \sum_{i=1}^{n} S_{i1}$ and $U_0 = \sum_{i=1}^{n} U_{i1}$. When formulating P_1, we have to write $\sum_{j \in N_i} x_{ij} \quad \leq \quad 1$ (instead of $\sum_{j \in N_i} x_{ij} = 1$) because the lightest items are assumed to be already inserted into the knapsack. Therefore, some or even all classes in Problem P_1 may contain no items, that is, it is allowed that $\sum_{j \in N_i} x_{ij} = 0$ for the optimal solution of Problem P_1.

Problem P_1:
Maximize $Z_1 = \sum_{i=1}^{n} \sum_{j \in N_i} (S_{ij} - S_{i1}) \ \ x_{ij}$
subject to $\sum_{i=1}^{n} \sum_{j \in N_i} (U_{ij} - U_{i1}) \ \ x_{ij} \leq (c - U_0)$,
$\sum_{j \in N_i} x_{ij} \quad \leq \quad 1, i = 1, ..., n$,
$x_{ij} = 0$ or 1, for $j \in N_i, i = 1, ..., n$.

P_2: The Truncated Relaxed MCK Problem

Problem P_2 is formulated from Problem P_1 by allowing a relaxation on the variable integrality condition: $0 \leq x_{ij} \leq 1$. Let Z_2 be the objective function of Problem P_2. The reason for introducing this problem is that its exact solution can be found in low computation time, which in turn, provides a good approximation solution to Problem P_1 and hence a good approximation solution to P_0. The algorithm for exact solving the Problem P_2 [12, 20, 16, 18] can be obtained by solving the following P_3 and P_4 problems.

P_3: The Relaxed MCK Problem on the Convex Hull

Given P_2, a convex hull of items in each class can be found [16]. The elements constituting the convex hull will be called *P-undominated* and denoted by (R_{ij}, H_{ij}) (this notion will be explained below in more detail).

Let us start by denoting $(S_{ij} - S_{i1})$ in P_2 by p_{ij} and $(U_{ij} - U_{i1})$ by w_{ij}.

Definition 1 (Sinha and Zoltners [20]). If two items r and s in the same class N_i in Problem P_2 satisfy that $p_{ir} \leq p_{is}$ and $w_{ir} \geq w_{is}$ then item r is said to be *dominated* by s.

Proposition 2 (Sinha and Zoltners [20]). In every optimal solution of P_3, $x_{is} = 0$, that is, the *dominated items* do not enter into the optimal solution.

Proposition 3 (Sinha and Zoltners [20]). If some items r, s, t from the same class N_i are such that $p_{ir} \leq p_{is} \leq p_{it}, w_{ir} \leq w_{is} \leq w_{it}$, and

$$\frac{(p_{is} - p_{ir})}{(w_{is} - w_{ir})} \leq \frac{(p_{it} - p_{is})}{(w_{it} - w_{is})}, \tag{3}$$

then $x_{is} = 0$ in every optimal solution of P_2.

The item $s \in N_i$ is called *P-dominated* [16]. In what follows, we exclude *P-dominated* points from each class N_i when solving the relaxed Problem P_3 to optimality. The items remaining after we excluded all the *P-dominated* points are called *P-undominated*. All these items belonging to the same class, if depicted as points in the two dimensional space (R, H), form the upper convex hull of the set N_i [16]. Note that R denotes energy savings and H denotes utilization.

The set of all *P-undominated* items may be found by examining all the items in each class N_i in an increasing order and according to Equation 3. Because of the ordering of the items, the upper convex hull can be found in $O(m\ log\ m)$ time [20]. Recall that $m = \sum_{i=1}^{n} n_i$. The obtained Multiple-Knapsack Problem on the Upper Convex Hull is denoted as Problem P_3.

Problem P_3**:**
Maximize $Z_3 = \sum_{i=1}^{n} \sum_{j \in N_i} R_{ij}\ \ y_{ij}$
subject to $\sum_{i=1}^{n} \sum_{j \in N_i} H_{ij}\ \ y_{ij} \le (c - U_0)$,
$\sum_{j \in N_i}\ \ y_{ij} \le 1, i = 1, ..., n$,
$0 \le y_{ij} \le 1$, for $j \in N_i, i = 1, ..., n$.

As described in [20], some items belonging to the class N_i (i.e., $y_{ij} = 1$) can be included into the solution entirely; they are called *integer variables.* On the other hand, some items may exceed the constraint: $\sum_{i=1}^{n} \sum_{j \in N_i} (H_{ij}\ \ y_{ij}) \le (c - U_0)$ and only part of it could be included into the solution. This items are called *fractional variables.*

P_4**: The Equivalent Knapsack Problem (EKP)**

The equivalent Knapsack Problem P_4 is constructed from P_3. In each class *slices*, or *increments* are defined as follows:

$$P_{ij} = (R_{ij} - R_{i,j-1});\ \ i = 1, \dots, n; j = 2, \dots, CH_i \tag{4}$$

$$W_{ij} = (H_{ij} - H_{i,j-1});\ \ i = 1, \dots, n; j = 2, \dots, CH_i \tag{5}$$

where CH_i is the number of the *P-undominated items* in the convex hull of class N_i. When solving the (continuous) Problem P_3, we may now discard the condition $\sum_{j \in N_i}\ \ x_{ij} \le 1, i = 1, ..., n$, and solve the problem of selecting *slices* in each class.

Problem P_4 :
Maximize $Z_4 = \sum_{i=1}^{n} \sum_{j \in N_i}\ \ P_{ij}\ \ z_{ij}$
subject to $\sum_{i=1}^{n} \sum_{j \in N_i} (W_{ij}\ \ z_{ij}) \le (c - U_0)$,
$0 \le z_{ij} \le 1$, for $j \in N_i$, $i = 1, ..., n$.

From the analysis of Problem P_4 [20, 12] it follows that, in all integer classes: if some variable is equal 1 (e.g., the variable is chosen) then all preceding variables are also 1; if some variable is equal zero (e.g., the variable is not chosen) then all subsequent variables are also zeros. From this fact the following important properties of Problem P_4 follow.

Property 1: *The sum of several* slices *in Problem P_4 correspond to a single item in Problem P_3, and in each class all the* slices *are numbered in the decreasing order of their ratios, $\frac{P_{ij}}{W_{ij}}$.*

Property 2: *There should not be a gap in a set of* slices *corresponding to a solution in any class.*

To exemplify this Property, let us consider the class N_j containing the *slices* r, s and t. According to Property 2, the following solutions are *valid*: $\{\}, \{r\}, \{r, s\}$ and $\{r, s, t\}$, while $\{s\}, \{t\}, \{r, t\}$ and $\{s, t\}$ are *invalid.* In particular, $\{r, t\}$ is invalid because *slice* s is not included, causing a gap in the solution.

6.3 Enhanced Greedy Algorithm

In order to solve the equivalent knapsack Problem P_4, we may collect all *slices* from all classes (following a decreasing order of their ratios, $\frac{P_{ij}}{W_{ij}}$) as candidates for including them into a single class: PW. With all *slices* in the single class PW, now the problem becomes the standard knapsack problem.

The main idea of the Standard Greedy Algorithm (SGA) for solving the standard knapsack is to insert the *slices*, $\{p_i, w_i\}$ (obtained from the single class PW) inside the available capacity of the knapsack $(c - U_0)$ in order of decreasing ratio $\frac{p_i}{w_i}$, until the knapsack capacity is completely full, or until no more *slices* can be included. If the knapsack is filled to its full capacity $(c - U_0)$ in the mentioned order, then this is the optimal solution. While inserting *slices* into the knapsack, one of them may not fit into the available capacity of the knapsack. This *slice* is called the *break-slice* [16], and its corresponding class is called the *break-class.*

Contrary to the solution proposed by Pisinger [18], our method does not consider *fractional items* to be part of the solution. Therefore, we will discard the *break-slices*, and consequently (following Property 2) all subsequent *slices* from the same *break-class.*

To the greedy scheme of [18] we add the following two rules.

- **Rule 1.** When computing the solution of P_4 take into account $Z_4^{'} = \{(p_{max}), \hat{Z}_4\}$, where $p_{max} = max\{p_i\}$ is the maximal energy saving item in the truncated MCKP P_2 and $\hat{Z}_4 = p_1 + p_2 + \ldots + p_{k-1}$ is the approximate solution obtained by the Standard Greedy Algorithm (SGA).
- **Rule 2.** After finding the *break-slice*, the remaining empty space is filled in by *slices* from the non-break classes in decreasing order of the ratios $\frac{p_i}{w_i}$.

The SGA algorithm is executed until the first *break-slice* is found. The Enhanced Greedy Algorithm (EGA) algorithm is executed for all *slices* in Problem P_4, which are included in order of their decreasing ratio $\frac{p_i}{w_i}$ $(i = 1, \ldots, \hat{n})$. According to Rule 2, *break-slices* are not considered to be part of the solution in the EGA algorithm. The SGA and EGA algorithms are illustrated in Figure 2.

6.4 Restoring the Solution from the EKP to the MCKP

An approximate solution to Problem P_4 is obtained as follows:

- SGA Algorithm: $Z_4^{'} = max\{p_{max}, (p_1 + p_2 + \ldots + p_{k-1})\}$
- EGA Algorithm: $Z_4^{''} = max\{p_{max}, (p_1 + p_2 + \ldots + p_{k-1} + \alpha)\}$

The term α is a possible increment caused by using Rule 2, that is, the profits of additional items from non-break classes.

```
Enhanced Greedy Algorithm: (EGA Algorithm)
input: a set of slices $p_j$ and $w_j$ from $P_4$ ordered by the ratio $\frac{p_i}{w_i}$
c: (size of the knapsack), $\hat{n}$:(number of items on Problem $P_4$)
output: $x_i$: (solution set);
$(p^*, u^*)$: (energy savings and utilization result)
begin
        $\bar{c} = (c - U_0)$; $p^* = 0$; $w^* = 0$;
        for j = 1 to $\hat{n}$ do
            if $w_j > \hat{c}$ then
                    $x_j = 0$; break-slice = j;
                    exit; (condition for SGA algorithm)
            else
                    $x_j = 1$; $\hat{c} = \hat{c} - w_j$;
                    $p^* = p^* + p_j$; $u^* = u^* + w_j$
end;
```

Fig. 2. Greedy Algorithms: SGA, EGA

The approximate solution to the Problem P_0 is defined as $Z_4 + S_0$. Recall that $S_0 = \sum_{i=1}^{n} S_{i1}$, are the elements truncated in Problem P_1.

From the definition of the *slice* (described in Equations 4 and 5) and Property 1, it follows that if several *slices*, (for example s, r and t in that order) belonging to the same class N_j are chosen to be part of the solution of the greedy algorithm, then the item corresponding to the *slice* t is considered to be part of the solution of P_0. On the other hand, if no *slice* is chosen from class N_j to be part of the solution, then the truncated item considered in Problem P_1 ($S_{j,1}$ and $U_{j,1}$) is chosen to be part of the solution.

The above criteria allows us to construct the corresponding items (speeds) from each class from Problem P_4 that are part of the solution of Problem P_0.

The solution from Problems P_1, P_2 and P_4 can be obtained in $O(m)$ time, while the EGA Algorithm obtains solutions in $O(m\ log\ m)$ time.

6.5 Scheduling the New Task

After the optimization procedure is executed, the Total Bandwidth Server (TBS) [14] will calculate the start time of the new task. It is well known that TBS Server provides low response times for handling aperiodic tasks. It is important to note that the newly arrived task may not be scheduled immediately at its arrival time because it may cause some missing deadlines. The resulting utilization, after executing the *optimization procedure*, may not be immediately subtracted from the total processor load because at the arrival time some tasks may already have delayed the execution of other tasks.

Finally, with the start time of the new task computed and the solution provided by the optimization procedure (the set of speeds for execution), the PORTS Server will schedule the new task in the system.

7 Simulation Experiments

The following simulation experiments have been designed to test the performance of the PORTS Server and its ability to achieve our optimality criteria using synthetic task sets. The goals in this simulation experiments are: (1) to measure the quality of the results over a large set of dynamic tasks that arrive and leave the system at arbitrary instants of time, and (2) to measure and compare the performance and run-time of our algorithms against known algorithms.

The algorithms used for comparison are: Dynamic Programming (DP) [16], Static Discrete Algorithm (SD) and the Optimal Discrete Algorithm OP(d) [1].

Each plot in the graphs represents the average of a set of 5000 task arrivals. The results shown in the graphs are compared with the SD Algorithm and the size of the knapsack used in the experiments is 1000 (100% of the load).

Each task has a life-time (r_i) that follows a uniform distribution between 30 and 200 instances (periods). At the end of its life-time, the task leaves the system. The period P_i of each task follow a uniform distribution between 1000 and 16000 time units, such that the LCM of the periods is 32000.

The arrival time of task T_{i+1} is computed by $a_{i+1} = \frac{P_{i+1} * r_{i+1}}{nt}$, where nt is the actual number of tasks in the system.

It is assumed that, for a given number of speeds, each speed level is computed proportionally between the maximum speed ($SP_{max} = 1.0$) and the minimum speed ($SP_{min} = 0.2$). For example, if there are 5 speed levels, the speed levels will be $\{1.0, 0.8, 0.6, 0.4, 0.2\}$. The utilization of task T_i under minimum speed, U_{in}, is chosen as a random variable with uniform distribution between 20% and 30%. C_i is computed by $C_i = U_{in} \cdot SP_{ij} \cdot P_i$. For each speed, utilization U_{ij} is computed by $U_{ij} = \frac{t_{ij}}{P_i}$, and $t_{ij} = \frac{C_i}{SP_{ij}}$.

The power functions for each task T_i used [11, 19, 21] are of the form $k_i \cdot S_i^{x_i}$, where k_i and x_i are random variables with uniform distributions between 2 and 10, 2 and 3 respectively. Then, the energy consumption for each task and each speed SP_j is computed by $E_{ij} = I \cdot (k_i \cdot SP_j^{x_i} \frac{C_i}{SP_j \cdot P_i})$, where I is a fixed interval, given by $I = LCM$. Finally, the input to our Optimization Problem P_0 is computed by Equation 1.

The performance of our algorithms is measured at each task arrival (and departure) according to the following metrics:

- **Percentage (%) of Energy Savings:** This metrics is computed as follows. The solution obtained (in terms of Energy Consumption) by each algorithm for all task give us the total energy consumption $E_{tot} = \sum_{i=1}^{n} E_i$. The solution provided by each algorithm is then compared with the solution obtained by algorithm SD, and the percentage of improvement is plotted in the graphs.
- **Run-Time:** This metrics denotes the execution time of each algorithm, which measures the physical time in microseconds, using a PC Intel 233 MHZ with 48MB of RAM and running on the Operating System Linux. The function used for the measurements is gettimeofday().

We show here two cases to demonstrate the performance of our algorithms. The first case (Figure 3), executes the simulations considering 10 speed levels, and the number of tasks is varied from 5 to 80. In the second case (Figure 4), the number of tasks is set to 30, and the speed level is varied from 3 to 60.

The results obtained by algorithm EGA (shown in Figure 3) vary from 95 to 99 near optimal solutions (when compared with the DP Algorithm), with % of energy savings ranging from 23 % to 25 %. While the SGA provide solutions from 92 to 96 near-optimal, with % of energy savings ranging from 19 % to 22 %. This results give an improvement of over 80% from the results obtained by the OP(d) algorithm. It is important to note that the continuous OP(c) algorithm was also simulated giving % of energy savings between 26 and 30.

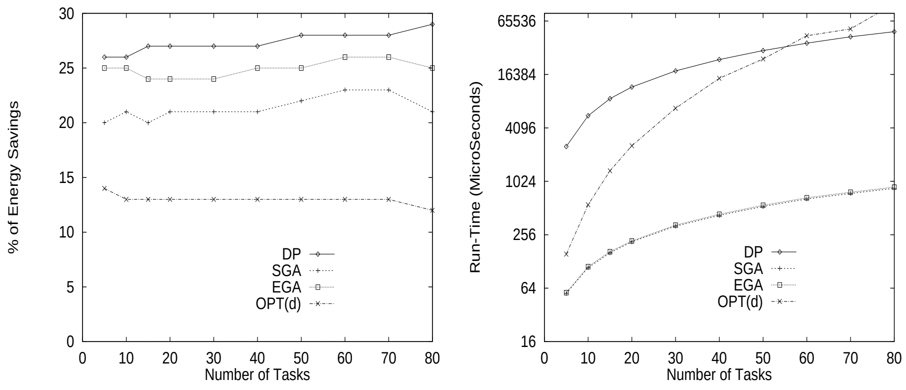

Fig. 3. % of Energy Savings and Run-Time (Microseconds)

The results shown in Figure 3 indicate the low cost of the enhanced greedy algorithms. For the SGA and EGA algorithms the run-time varies from 56 to 853 microseconds. Note the large difference in run-time obtained by the EGA algorithms when compared with the DP and the OP(d) algorithms. For our simulation settings, the OP(d) algorithm varies from 155 to 102500 microseconds, and the DP algorithm varies from 2529 to 49653 microseconds.

The results shown in Figure 4 indicate how important is to consider an appropriate number of speed levels for achieving a high percentage of energy savings. As shown in Figure 4, under low number of speed levels, between 3 and 30, the EGA algorithm gives better performance than the OPT(d) algorithm. However, for more than 30 speed levels OPT(d) algorithm outperforms the EGA algorithm. For this experiment the run-time computed (shown in Figure 4), indicate that the OPT(d) algorithm has very little sensibility to the number of speed levels (i.e., the run-time of the OPT(d) algorithm varied from 6900 to 7100 microseconds). In contrast, our Greedy Algorithms increased their run-time with higher number of speed levels. For this experiments, the run-time of the Greedy Algorithms varied from 99 to 1800 microseconds.

Further tests were conducted (increasing the number of speed levels) to conclude that both the EGA algorithm and the OPT(d) algorithms have similar run-time, when the number of speed levels is reaching 100.

The results obtained in our simulations indicate that the Enhanced Greedy Algorithms are a low cost and effective solutions for scheduling power-aware real-time tasks with discrete speeds.

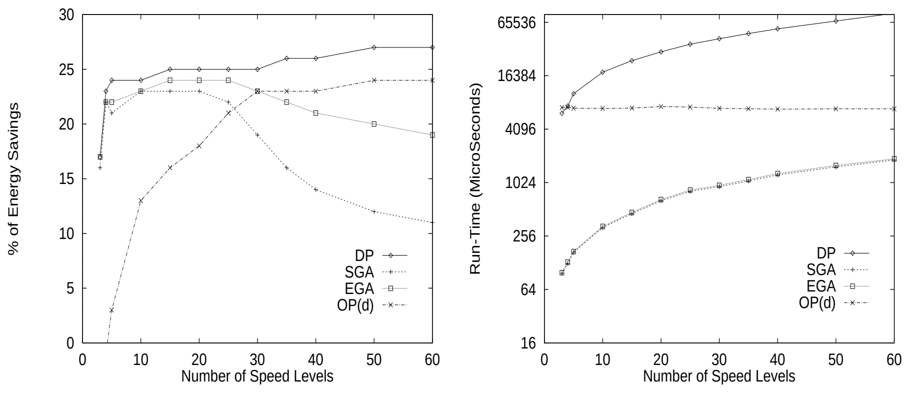

Fig. 4. % of Energy Savings and Run-Time (Microseconds)

8 Conclusions

In this paper we proposed a power optimization method for a real-time application running on a variable speed processor with discrete speeds. The solution proposed is based on the use of a Power-Optimized Real-Time Scheduling Server (PORTS) which is comprised of two parts (a) a feasibility test, for testing the admission of new dynamic tasks arriving in the system, and (b) an optimization procedure used for computing the levels of speed of each tasks in the system, such that energy savings of the system is maximized. The process of selecting levels of voltage/speed for each tasks while meeting the optimality criteria requires the exploration of a potentially large number of combinations, which is infeasible to be done on-line. The PORTS Server finds near-optimal solutions at low cost by using approximate solutions to the knapsack problem.

Our simulation results show that our PORTS Server has low overhead, and most importantly generates near-optimal solutions for the scheduling of real-time systems running on variable speed processors.

We will extend the PORTS Server with algorithms for multiple processors and for real-time tasks with precedence and resource constraints.

References

1. H. Aydin, R. Melhem, D. Mosse, P. Mejia. “Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics”. *EuroMicro Conference on Real-Time Systems*, June 2001.
2. H. Aydin, R. Melhem, D. Mosse, P. Mejia. “Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems”. *IEEE Real-Time Systems Symposium*, Dec. 2001.

3. T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System", IEEE J. of Solid-State Circuits, Vol. 35, No. 11, Nov. 2000.
4. F. Gruian, K. Kuchcinski. "LEneS:Task Scheduling for Low Energy Systems Using Variable Supply Voltage Processors". In *Proc. Asia South Pacific - DAC Conference 2001*, June 2001.
5. I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava. "Power Optimization of Variable Voltage Core-Based Systems". In *Design Automation Conference*, 1998.
6. I. Hong, M. Potkonjak and M. B. Srivastava. "On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor". In *Computer-Aided Design (ICCAD)'98*, 1998.
7. I. Hong, G. Qu, M. Potkonjak and M. Srivastava. "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors". In *Proc. of 19th IEEE Real-Time Systems Symposium*, Madrid, December 1998.
8. Intel, Microsoft, Compaq, Phoenix and Toshiba. "ACPI Specification", *developer.intel.com/technology/IAPC/tech.*
9. Intel StrongARM SA-1100 microprocessor developer's manual.
10. T. Ishihara and H. Yasuura. "Voltage Scheduling Problem for Dynamically Varying Voltage Processors", *In Proc. Int'l Symposium on Low Power Electronics and Design*, 1998.
11. C. M. Krishna and Y. H. Lee. "Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems". In *Proc. of the IEEE Real-Time Technology and Applications Symposium*, 2000.
12. E. Lawler. "Fast Approximation Algorithms for Knapsack Problems". *Mathematics of Operations Research*, Nov. 1979.
13. C.L. Liu, J. Layland. "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments", *J. ACM*, 20(1). Jan. 1973.
14. G. Lipari, G. Buttazzo. "Schedulability Analysis of Periodic and Aperiodic Tasks with Resource Constraints", *J. of Systems Architecture*, (46). 2000.
15. J.R. Lorch, A.J. Smith. "Improving Dynamic Voltage Scaling Algorithms with PACE". In *Proc. of ACM SIGMETRICS Conference* Cambridge, MA, June 2001.
16. S. Martello and P. Toth. "Knapsack Problems. Algorithms and Computer Implementations". *Wiley*, 1990.
17. D. Mosse, H. Aydin, B. Childers, R. Melhem. "Compiler Assisted Dynamic Power-Aware Scheduling for Real-Time Applications". *In Workshop on Compiler and Operating Systems for Low Power COLP'00* October, 2000.
18. D. Pisinger. "A Minimal Algorithm for the Multiple-Choice Knapsack Problem", *European Journal of Operational Research*, 83. 1995.
19. Y. Shin and K. Choi. "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems". In *Proc. of the Design Automation Conference*. 1999.
20. P. Sinha, A. Zoltners. "The Multiple Choice Knapsack Problem". *Operations Research*, May-June 1979.
21. V. Swaminathan, K. Chakrabarty. "Investigating the Effect of Voltage-Switching on Low-Energy Task Scheduling in Hard Real-Time Systems". In *Proc. Asia South Pacific - DAC Conference 2001.*
22. www.arm.com
23. www.transmeta.com
24. F. Yao, A. Demers, S. Shenker. "A Scheduling Model for Reduced CPU Energy". *IEEE Annual Foundations of Computer Science*, 1995.