Power-Optimized Scheduling Server for Real-Time Tasks

Pedro Mejia-Alvarez *, Eugene Levner†, Daniel Mossé ‡

Abstract

In this paper we propose a novel scheduling framework for a dynamic real-time environment with energy constraints. This framework dynamically adjusts the CPU voltage/frequency so that no task in the system misses its deadline and the total energy savings of the system are maximized. In this paper we consider only realistic, discrete-level speeds.

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 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 is able to generate power-aware scheduling solutions with near-optimal performance.

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, PDAs and other mobile and embedded computing systems simply because it leads to extended battery lifetime. Further, the need for power-efficient designs is not solely associated with portable computing systems. Power dissipation has become a design constraint in virtually every type of computing system including desktop computers, network routers and switches, set-top entertainment systems and the most performance-hungry computer servers.

The problem of reducing and managing energy consumption was 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) [9] for energy management of laptops, desktops and servers that allow several modes of operation, several "sleep" levels, and the ability to turn off some parts of the computer (e.g., the disk) after a preset period of inactivity. More recently, *variable voltage scheduling* (VVS), has been proposed as an alternative to energy management. In VVS the voltage and frequency (hence, the CPU speed) is adjusted dynamically. Because the power consumption is linearly dependent on the time and quadratically dependent on voltage, reducing the frequency and voltage (we call this *reducing the speed*), operations will consume less energy, but take longer to complete.

In this paper, the VVS problem in real-time systems is to assign appropriate speeds (from a set of discrete speeds) to a set of dynamically arriving and departing periodic tasks, such that no task misses its predefined deadline while the total energy savings in the system is maximized. 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 formulated as a multiple-choice knapsack problem (MCKP) with binary variables [18].

In order to cope with the high computation costs of the dynamic real-time environment, we have developed a low-cost power-aware scheduling scheme. Our Power-Optimized Real-Time Scheduling (PORTS) Server 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 prob-

*CINVESTAV-IPN, Sección de Computación, AV. IPN. 2508, México. DF. pmejia@cs.cinvestav.mx

†Holon Academic Institute of Technology, Department of Computer Science, 52 Golomb ST, Holon 58102 Israel. levner@hait.ac.il

‡Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260. mosse@cs.pitt.edu. This author was supported in part by DARPA PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736 and through NSF-ITR medium grant.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

lem into a standard knapsack problem, (c) a *greedy heuristic algorithm* 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 PORTS methodology provides a novel approach to real-time scheduling which yields 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 known algorithms.

The rest 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 are defined. In Section 4, the power-optimized scheduling is formulated as an optimization problem. In Section 5, the Power-Optimized Real-Time Scheduling (PORTS) Server is described and in Section 6 we analyze the merit of the optimization procedure of the PORTS server using an example, and compare its performance against other known algorithms. In Section 7, simulation results are presented to show the performance of the PORTS Server. Finally, Section 8 presents concluding remarks.

2 Related Work

In this paper we only address the issue of reducing power consumption of processors through dynamically changing the speed (i.e., voltage and frequency) of a processor. This technique can be classified as *static* and *dynamic*. Static techniques, such as static scheduling, compilation for low power [19] and synthesis of systems-on-a-chip [8], are applied at design time. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving dynamically arriving real-time tasks or variable workloads.

Static (or off-line) scheduling methods to reduce power consumption in real-time systems were proposed in [28, 11, 6]. 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 [7]. Non-preemptive power-aware scheduling is investigated in [6]. Recent work on VVS includes the exploitation of idle intervals in the context of the Rate Monotonic and Earliest Deadline First (EDF) scheduling frameworks [23, 13, 2, 17]. Following the same VVS technique, other work [20, 21] consider the fact that some real-time tasks not always execute their worst-case execution times, and have the ability to dynamically reclaim unused computation time to obtain additional energy savings.

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, 5]. A recent survey of such methods, considering continuous voltage has shown how many different algorithms behave [24].

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 [27] (200–700 MHz in 33 MHz steps, 1.1–1.6V); (b) the ARM7D processor [10] (20 or 33MHz, 5V or 3.3V); (c) the Intel StrongARM SA1100 processor [10] (59–221 MHz in 14.7 MHz Steps); and (d) the Intel XScale (150MHz–1GHz).

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) [15] 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) video-on-demand 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. These types of system can be found in communication satellites, which have an extreme need for extending power management (for instance, a 10% increase in battery life for a satellite with a 5-year estimated lifetime would make the satellite functional –and profitable– for an extra 1/2 year).

We denote by C_i the number of processor cycles required by T_i in the worst-case. Under a constant speed V_i (given in cycles per second), the execution time of the task is $t_i = \frac{C_i}{V_i}$. 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 a task is processor load that a task demands for execution: $U_i = \frac{t_i}{P_i}$ (or $\frac{C_i}{V_i P_i}$). According to EDF, a set of tasks are feasible (no tasks misses its deadline) if $\sum U_i \leq 100\%$. Although we assume EDF in this paper, the scheme can be easily extended to fixed-priority schedulers [15, 12].

We assume that the CPU speed can be changed at *discrete* levels between a minimum speed V_{min} (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed V_{max}.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

V_{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 $g_i(V)$, 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 $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(V)dt$. 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 two additional assumptions. First, the time overhead associated with voltage switching is negligible, that is, the voltage change overhead can be incorporated in the worst-case workload of each task. Second, different tasks have different power consumptions, that is, the power dissipation is dependent on the nature of the running software of each task in the system. For example, a task may use more memory or the floating point unit more than others, and a task may ship the tasks to specialized processors (e.g., DSPs, microcontrollers, or FPGAs).

4 Formulation of the Problem

The problem we address can be formulated as follows. Each time a new task T_i arrives or leaves the system, 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; thus, the solution should have low computation complexity.

4.1 The Optimization Problem

We define a set of speeds, N_i, for each task T_i. Each level of speed $j \in N_i$ has a Energy Saving computed by

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

where E_{ij} is the energy consumed by task T_i executing at speed j ($j = 1$ means maximum speed[1].

Furthermore, each task running at speed V_{ij}, will have utilization $U_{ij} = \frac{C_i}{V_{ij} \cdot P_i}$. It is assumed that the items $j \in N_i$ are arranged in non-decreasing order, so that S_{i1} and U_{i1} are the items with the smallest values.

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 different tasks arrivals a_k and a_{k+1}. S^k denotes the amount of energy savings accrued by all the tasks in the system during $a_{k+1} - a_k$, that is, $S^k = \sum_{i=1}^{n} S_i^k$.

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 (i.e., 100%). That is, Problem P_0 is defined as follows.

$$\begin{aligned} \textbf{maximize} \quad & Z_0 = \sum_{i=1}^{n} \sum_{j \in N_i} S_{ij} \; x_{ij} \\ \textbf{subject to} \quad & \sum_{i=1}^{n} \sum_{j \in N_i} U_{ij} \; x_{ij} \leq 100\% \\ & \sum_{j \in N_i} x_{ij} = 1, \; i = 1, ..., n \end{aligned}$$

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

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 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 [18]. However, the MCKP is known to be NP-hard [18] which implies that it is very unlikely to design a fast (polynomial-time) exact method for its solution. From a practical point of view this means that exact methods, such as dynamic programming [18], Lagrange multipliers [1], mixed-integer linear programming [26] and enumeration schemes [7], do not satisfy realistic temporal requirements for solving any reasonable size MCKP (that is, cannot generate a solution with milliseconds).

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 policy (EDF [15]). 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 proposed method consists of five basic parts, or stages, as illustrated in Figure 1, and described in detail in the following subsections.

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]Note that, $S_{i1} = E_{i1} - E_{i1} = 0$; that is executing at maximum speed yields no energy savings.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

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 solution of the standard 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 [16] 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 does not need to be executed since the system utilization is decreased when a task terminates.

5.1 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 criterion 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.

5.2 Reduction Scheme from MCKP to the Standard KP

Our approximation algorithm is based on the reduction of the MCKP to the equivalent KP using the convex hull concept [18]. 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 (i.e., the item with the minimum U_{ij} value) 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 $S_{min} = \sum_{i=1}^{n} S_{i1}$ and $U_{min} = \sum_{i=1}^{n} U_{i1}$. When formulating P_1, we have to consider the new capacity of the system equal to $(c - U_{min})$ and we have to write $\sum_{j \in N_i} x_{ij} \leq 1$ (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 may happen that $\sum_{j \in N_i} x_{ij} = 0$ for the optimal solution of Problem P_1. Notice that $c = 1.0$ (or $c = 100\%$) denotes the maximum capacity of the system.

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

P_2: The Truncated Relaxed MCK Problem

Problem P_2 is constructed from Problem P_1 by allowing a relaxation on the variable integrality condition: $0 \leq x_{ij} \leq 1$, in other words, x_{ij} values are not necessarily 0 or 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 [25]. Problem P_2 can be solved by the following P_3 and P_4 problems [14, 25, 18, 22].

P_3: The Relaxed MCK Problem on the Convex Hull

Given P_2, a convex hull of items in each class can be found [18]. 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 the savings without the lightest item in P_2 by $s_{ij} = S_{ij} - S_{i1}$ and analogously denote the utilization by $u_{ij} = U_{ij} - U_{i1}$.

The following Dominance Criterion (Sinha and Zoltners [25]) is applied to reduce the set of items in the convex hull.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

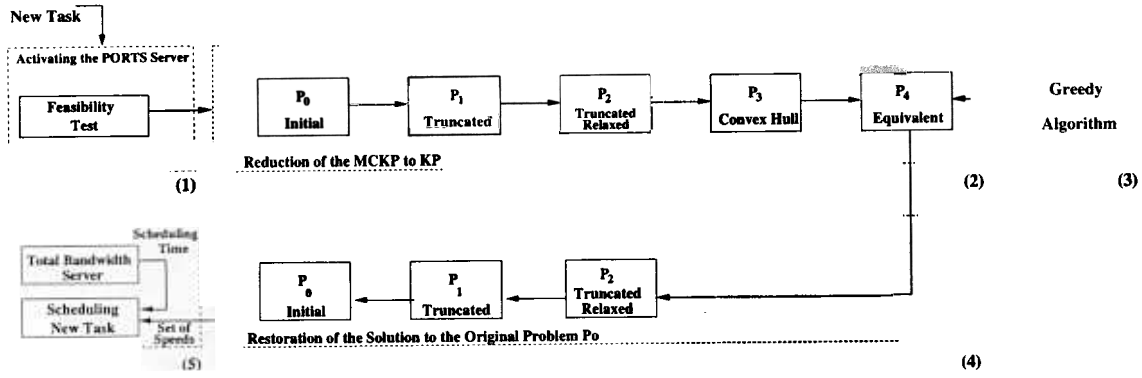

Figure 1. Methodology for Handling Power-Aware Real-Time Tasks

Dominance Criterion 1. If two items r and q in the same class N_i in Problem P_2 satisfy $s_{ir} \leq s_{iq}$ and $u_{ir} \geq u_{iq}$ then item r is said to be *dominated* by item q. In every optimal solution of P_3, $x_{ir} = 0$, that is, the *dominated items* do not enter into the optimal solution.

Dominance Criterion 2. If some items r, q, t from the same class N_i are such that $s_{ir} \leq s_{iq} \leq s_{it}, u_{ir} \leq u_{iq} \leq u_{it}$, and

$$\frac{(s_{iq} - s_{ir})}{(u_{iq} - u_{ir})} \leq \frac{(s_{it} - s_{iq})}{(u_{it} - u_{iq})} \quad (2)$$

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

The item $q \in N_i$, depicted in Figure 2, is called *P-dominated*[25]. In what follows, we exclude *P-dominated* items from each class N_i when solving the relaxed Problem P_3 to optimality. The items remaining after we excluded all the *P-dominated* items are called *P-undominated*. All the *P-undominated* items belonging to the same class, if depicted as points in the two dimensional space, form the upper convex hull of the set N_i [18], and denote the new set of P-undominated items $\{(R_{ij}, H_{ij})\}$, as illustrated in Figure 2. Note that R denotes energy savings and H denotes utilization.

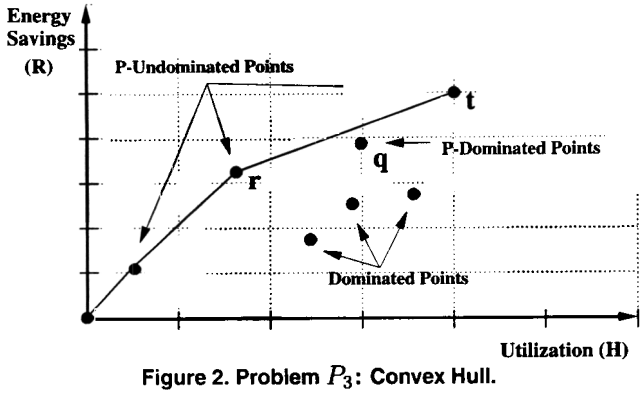

Figure 2. Problem P_3: Convex Hull.

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 (2). Because of the ordering of the items, the upper convex hull can be found in $O(m \log m)$ time [25]. 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 $\sum_{i=1}^{n} \sum_{j \in N_i} H_{ij}\ y_{ij} \leq (c - U_{min})$,
$\sum_{j \in N_i}\ y_{ij} \leq 1, i = 1, ..., n,$
$0 \leq y_{ij} \leq 1$, for $j \in N_i, i = 1, ..., n.$

As described in [25], 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: $\sum_{i=1}^{n} \sum_{j \in N_i} (H_{ij}\ y_{ij}) \leq (c - U_{min})$ and only part of it could be included into the solution. These 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:

$$P_{ij} = (R_{ij} - R_{i,j-1});\ \ i = 1, \ldots, n; j = 2, \ldots, CH_i \quad (3)$$
$$W_{ij} = (H_{ij} - H_{i,j-1});\ \ i = 1, \ldots, n; j = 2, \ldots, CH_i \quad (4)$$

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[2] the condition $\sum_{j \in N_i}\ y_{ij} \leq 1, i = 1, ..., n$, 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 $\sum_{i=1}^{n} \sum_{j \in N_i} (W_{ij}\ z_{ij}) \leq (c - U_{min})$,
$0 \leq z_{ij} \leq 1$, for $j \in N_i, i = 1, ..., n.$

[2]This condition is discarded because after Problem P_4 we will include all items from all tasks into a single class in the Enhanced Greedy Algorithm discussed in the next Section.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

From the analysis of Problem P_4 [25, 14] 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 combination of all items (slices) from class N_i ($i = 1, ..., n$) yielding an optimal solution to Problem P_4, corresponds to a specific item to be chosen from class N_i in Problem P_3; namely, if k denotes this specific item, then $\sum_{\{j \in N_i\}}(P_{ij}\, x_{ij}) = R_{ik}$ and $\sum_{\{j \in N_i\}}(W_{ij}\, x_{ij}) = H_{ik}$. 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 Property 1, assume that $P_{1,1} = 1642$, $P_{1,2} = 2764$, and $P_{1,3} = 2074$ are the slices from class N_1. This optimal solution of Problem P_4 correspond to $R_{1,3} = 6480$ from Problem P_3. To exemplify Property 2, let us consider the class N_j containing the *slices* r, q and t. According to Property 2, the following solutions are *valid*: $\{\}, \{r\}, \{r, q\}$ and $\{r, q, t\}$, while $\{q\}, \{t\}, \{r, t\}$ and $\{q, t\}$ are *invalid*. Note that, $\{r, t\}$ is invalid because *slice* q is not included, causing a *gap* in the solution.

5.3 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 $(c - U_{min})$ 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 exactly to its full capacity $(c - U_{min})$ 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* [18], and its corresponding class is called the *break-class.*

Contrary to the solution proposed by Pisinger [22], 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 remaining *slices* from the same *break-class.*

To the greedy scheme of [22] 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 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, in contrast to SGA, does not stop when the first break-slice is found; it is executed for all remaining *slices* in the single class PW. 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 3.

```
1   Enhanced Greedy Algorithm:  (EGA Algorithm)
2   input:  a set of slices p_j and w_j from P_4
                          ordered by the ratio p_i/w_i
3:  ĉ = (c − U_min), n̂:(number of items on P_4)
4:  output:  x_i, (p*, w*):  (solution set);
5:  begin
6:      p* = 0;  w* = 0;
7:      for j = 1 to n̂ do
8:          if w_j > ĉ then
9:              x_j = 0; break-slice = j;
10             remove the remaining slices from
                    the break class (Property 2)
11:            exit; (condition for SGA algorithm)
12:         else
13:             x_j = 1;  ĉ = ĉ − w_j;
14:             p* = p* + p_j;  w* = w* + w_j
15: end;
```

Figure 3. Greedy Algorithms: SGA, EGA

5.4 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.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

```
1:  Optimization Procedure:
2:  input:  set of S_ij (energy savings)and U_ij (utilization) from problem P_0
3:            c:  (size of the knapsack)
4:  output:  x_ij , (p*, u*) :          (solution:  vector, energy savings and utilization)
5:  begin
6:        S_min = Σ_{i=1}^{n} S_i1 ; U_min = Σ_{i=1}^{n} U_i1 ; (Savings and Utilization at maximum speed)
7:        for j = 1 to n do
8:        begin
9:  P1          s_ij = S_ij − S_i1 ; u_ij = U_ij − U_i1 ; ĉ = (c − U_min);
10: P3          for (all items in N_i)                  (items r, q and t belong to N_i)
11:                 if (s_ir ≤ s_iq and u_ir ≥ u_iq) then
12:                   remove item 'r' from class N_i;        (Dominance Criterion 1)
13:                 if ((s_iq − s_ir)/(u_iq − u_ir) ≤ (s_it − s_iq)/(u_it − u_iq)) then
14:                   remove item 'q' from class N_i;        (Dominance Criterion 2)
15:             for (all P-undominated items in N_i) do
16:                 R_ij = s_ij ; H_ij = u_ij ;          (Convex Hull with P-undominated items)
17: P4:         P_ij = (R_ij − R_{i,j−1}); W_ij = (H_ij − H_{i,j−1}); j = 2, ..., CH_i          (slices)
18:       end
19:       Insert all (P,W) items in order of P/W into array PW
20:       Execute the Greedy Algorithm with PW as input.        (output:  x_ij, p*, w*)
21:       P = S_min + p* ; W = U_min + w* ;     (energy savings and utilization solution)
22: end
```

Figure 4. Optimization Procedure

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

From the definition of the *slice* (described in Equations 3 and 4) 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_{j1} and U_{j1}) 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 (lines 6 to 18 from Figure 4), while the EGA Algorithm obtains solutions in $O(m\ log\ m)$ time (lines 19 to 22 from Figure 4).

The Algorithm that describes the Optimization Procedure defined in Section 5, is illustrated in Figure 4. The Reduction Algorithm (Problems P_1, P_3 and P_4), is executed in lines 7 to 19. The Approximation Algorithm is executed in lines 20 and 21, and the Restoration Algorithm is executed in lines 22 and 23.

5.5 Scheduling the New Task

After the optimization procedure is executed, the Total Bandwidth Server (TBS) [16] will calculate the start time of the new task. It is well known that TBS Server provides low response times for handling aperiodic tasks. As described in [4], the TBS Server assigns a deadline $d_{TBS} = max(t_a, d_{a-1}) + \frac{C_a}{U_s}$ to new aperiodic requests τ_a that arrive at time $t = t_a$. In our framework, C_a denotes the computation time of the new task and U_s denotes the server utilization factor. U_s is computed in our framework as $U_s = 1 - \sum_i U_i$. If the deadline of the new task obtained by the TBS is greater than the actual deadline of the new task, $d_{TBS} > d_a$, then the scheduling time of the new task is modified to $t_a^* = d_{TBS} - t_a$. On the other hand, if $d_a > d_{TBS}$ the new task is scheduled at its arrival time (t_a). In any case, old tasks may be preempted by the new task.

Finally, with the start time of the new task computed by the TBS server, 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 following the EDF scheduling policy.

6 Example

The purpose of this section is to illustrate the execution of our algorithms and to compare their performance against several other known algorithms:

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

- *Dynamic Programming (DP):* This algorithm was designed to solve to optimality the MCKP problem [18].
- *Maximum Speed (MS).* In this algorithm, the processor is set to its maximum speed (in this case, maximum speed = 1.0).
- *Static Continuous (SC) and Static Discrete (SD) Algorithms.* In the SC algorithm [1], the processor speed for all tasks is set to the system utilization (i.e., $V_i = \sum U_{i1}$). The static discrete solution uses the continuous solution (SC) and approximate its results. For example, if the speed levels of the processor are 1.0, 0.75, 0.5 and 0.25, and the utilization of the task set is 0.6, then the speeds of all tasks are set to 0.75.
- *Optimal Continuous OP(c) and Optimal Discrete Algorithms OP(d).* The continuous algorithm OP(c) [1] considers tasks with different power characteristics and provides a continuous solution. The discrete solution OP(d) approximates the continuous solution using discrete levels of speed, in the same way that SD approximates SC.

Consider the real-time workload described in Table 1. The value of c used in this example is 1000 (this value denotes 100% of the size of the knapsack). The workload described was computed assuming maximum speed levels for each task, obtaining a total load of $\sum_i U_i = 60\%$. In this example, a power consumption function $g_i(V) = k \cdot V^3$ is considered, where k is shown in Table 1. Table 2 shows the energy consumption E_{ij}, energy savings S_{ij} and utilization U_{ij} for the set of speeds of all tasks, $N_i = \{1.0, 0.9, 0.7, 0.5, 0.3\}$.

Tasks	C_i	P_i	U_i	k
Task 1	216	1600	0.135	2
Task 2	228	2000	0.114	2
Task 3	300	2000	0.150	8
Task 4	1551	8000	0.194	4

Table 1. Real-Time Workload and Parameter k

We will measure the energy consumption for a period of 32,000 time units, which is larger than the least common multiple of all the task periods P_i. Since the energy consumed by task T_i executing continuously in the interval of time I is $I \cdot g_i(V_i)$, the total energy consumed during the interval is $I \cdot \sum(g_i(V_i)\ \frac{C_i}{V_i P_i})$. This is because $\frac{I}{P_i}$ is the number of instances of task T_i within the interval I, and $\frac{C_i}{V_i}$ is the length of each instance.

Table 3 show the results of the truncation procedure described in Problem P_1. This Table is constructed with elements from table 2, removing the elements with minimum U_{ij} value.

Note that, the elements S_{11} and U_{11} in Table 3, Problem P_1, are computed by $S_{11} = Sij - S_{min} = 1642 - 0 =$ 1642 and $U_{11} = U_{ij} - U_{min} = 150 - 135 = 15$. After building Table 3 all minimum items from each task are included into the knapsack. So, the remaining size of the knapsack is: $\hat{c} = (1000 - U_{min}) = 408$.

		Speed Levels				
		1.0	0.9	0.7	0.5	0.3
	E_{ij}	8640	6998	4234	2160	778
	S_{ij}	0	1642	4406	6480	7862
Task 1	U_{ij}	135	150	192	270	450
	E_{ij}	7296	5910	3575	1824	657
	S_{ij}	0	1386	3721	5472	6639
Task 2	U_{ij}	114	126	162	228	380
	E_{ij}	38400	31104	18816	9600	3456
	S_{ij}	0	7296	19584	28800	34944
Task 3	U_{ij}	150	166	214	300	500
	E_{ij}	24816	20101	12160	6204	2233
	S_{ij}	0	4715	12656	18612	22583
Task 4	U_{ij}	193	215	276	387	646

Table 2. Energy Consumption, Savings and Utilization

The result of the slicing procedure of Problem P_4 is shown in the right hand side of Table 3. Following the slicing procedure of Problem P_4, it is easy to verify that $P_{14} = S_{14} - S_{13} = 7862 - 6480 = 1382$ and $W_{14} = U_{14} - U_{13} = 315 - 135 = 180$. Finally, ordering the elements in Table 3 by decreasing *energy/utilization* ratio produces the array PW, which is shown in Table 4.

Following the execution of the greedy algorithms (see Table 4) it is easy to verify that the solution for the greedy algorithms is,

SGA Algorithm: $S = 49583$, and $U = 340$.
EGA Algorithm: $S = 51334$, and $U = 406$.

Note that *slice* (5956,111) in PW, is the *break-slice* (B-S). According to our Restoring algorithm, the minimal elements from each class that were truncated in Problem P_1 must be added to the solution. Then $S_{min} = \sum_{i=1}^{4} S_{i,1} = 0$ and $U_{min} = \sum_{i=1}^{4} U_{i,1} = 592$, must be added to the solution. Therefore, the solution to the MCKP Problem P_0 is as follows.

SGA Algorithm: $S = 49583$, $U = 932$, and solution vector = [3,3,4,3]
EGA Algorithm: $S = 51334$, $U = 998$, and solution vector = [3,4,4,3]

In Table 5, the energy consumption solution for several algorithms is shown, along with the solution vector, the run-time (measured in microseconds on a PII-233 MHz) and the percentage of energy savings normalized to the SD algorithm.

Note that the OP(c) algorithm provides the best energy consumption results. However, because of the fact that we are using discrete levels of speed, its corresponding discrete solution (obtained by the OP(d) Algorithm), gives

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

	Problem P_1					Problem P_4				
	S_{ij}	1642	4406	6480	7862	P_{ij}	1642	2764	2074	1382
Task 1	U_{ij}	15	57	135	315	W_{ij}	15	42	78	180
	S_{ij}	1386	3721	5472	6639	P_{ij}	1386	2335	1751	1167
Task 2	U_{ij}	12	48	114	266	W_{ij}	12	36	66	152
	S_{ij}	7296	19584	28800	34944	P_{ij}	7296	12288	9216	6144
Task 3	U_{ij}	16	64	150	350	W_{ij}	16	48	86	200
	S_{ij}	4715	12656	18612	22583	P_{ij}	4715	7941	5956	3971
Task 4	U_{ij}	22	83	194	453	W_{ij}	22	61	111	259

Table 3. Result from Problems P1 and P4

P_{ij}	7296	12288	4715	7941	1386	1642	9216	2764	2335	5956	6144	2074	1751
W_{ij}	16	48	22	63	12	15	86	42	36	111	200	78	66
$\sum S$	7296	19584	24299	32240	33626	35268	44484	47248	49583				51334
$\sum Load$	16	64	86	149	161	176	262	304	340				406
$\frac{Included}{Not-Included}$	I	I	I	I	I	I	I	I	I	B-S	N-I	N-I	I

Table 4. Array PW: Greedy and Enhanced Greedy Algorithms

a performance lower than that of our Greedy Algorithms. Note that, the maximum speed algorithm achieves more than double (e.g., 104%) of energy savings than that of the SD algorithm. The solution vectors of algorithms OP(c) and SD are shown in Table 5 as "xx" because they do not yield discrete levels of speed. In the SC algorithm the speed is set to the value of the utilization (0.6).

Algorithm	Energy	Solution Vector	Run Time	% Savings
DP	26337	[2,3,4,4]	3980	33 %
MS	79152	[1,1,1,1]	586	- 104 %
SC	28495	[x,x,x,x]	586	27 %
SD	38784	[3,3,3,3]	586	0 %
OP(c) [1]	25711	[x,x,x,x]	1106	34 %
OP(d)	34668	[2,2,4,3]	709	11 %
SGA	29569	[3,3,4,3]	47	24 %
EGA	27818	[3,4,4,3]	58	29 %

Table 5. Results for Different Algorithms

Note that for this example, the Convex Hull of Problem P_3 will give no P-dominated elements. This result is obtained because the speeds for each task are the same on each N_i and because on each N_i there is a constant increase in the value of each item. Recall that in this example, the set of speeds N_i for all tasks are 0.3, 0.5, 0.7, 0.9 and 1.0. This arrangement on the set of speeds produces a Convex Hull with only P-Undominated points (see Figure 2) and may lead to a simplified algorithm without the convex hull. In this "special case", we may remove lines 10–16 from the Optimization Procedure in Figure 4.

According to the results obtained in this example, we can verify that our Greedy Algorithms can obtain results which are close to the optimal (discrete) solution obtained by the Dynamic Programming Algorithm.

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) [18], 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 $c = 1000$ (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 at the time T_{i+1} was generated.

For a given number of speeds, each speed level is computed proportionally between the maximum speed ($V_{max} = 1.0$) and the minimum speed ($V_{min} = 0.2$). 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_{min}, is chosen as a random variable with uniform distribution between 20% and 30%. C_i is computed by $C_i = U_{min} \cdot V_{ij} \cdot P_i$. For each speed, utilization U_{ij} is computed by $U_{ij} = \frac{t_{ij}}{P_i}$, and $t_{ij} = \frac{C_i}{V_{ij}}$.

The power functions for each task T_i used [13, 23, 26] 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 V_j is computed by $E_{ij} = I \cdot (k_i \cdot V_j^{x_i} \frac{C_i}{V_j \cdot P_i})$, where I is a fixed interval, given by $I = LCM$.

The PORTS server execute at the speed of the *current* executing task, and 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 ($\%ES$):** The solution obtained (in terms of Energy Consumption) by each algorithm for all task gives 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. The results shown in the graphs represent the average ($\bar{X} = (\sum_{i=1}^{5000} \%ES)/5000)$) or *sample mean* of a set of 5000 experiments (each for every task arrival). On each result of $\%ES$ we also compute their corresponding confidence intervals (for the mean) assuming a confidence level of 95%.
- **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().
- **Rejection Ratio (RR):** This metric denotes the percentage of tasks rejected from execution. A new task is rejected if it produces an overload. That is, a new task is rejected if the following condition is met $\sum_i U_i > 100\%$, while setting all tasks at maximum speed.

We demonstrate the performance of our algorithms with two simulation cases. The first case (Figure 5), considers 10 speed levels, and the number of tasks is varied from 5 to 80. In the second case (Figure 6), we show the influence of the granularity of the speed changing steps: 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 5) vary from 95% to 99% of the DP Algorithm (we use DP as our *optimal* Algorithm), with % of energy savings ranging from 23% to 25%.

The SGA performs from 92% to 96% of optimal, with energy savings ranging from 19% to 22%. This results show an improvement of over 80% from the results obtained by the OP(d) algorithm. It is important to note that, although unrealistic due to the lack of discrete speed levels, the continuous OP(c) algorithm was also simulated to give us an upper bound on the energy savings; it yields between 26% and 30% energy savings, showing that our heuristics perform very well with respect to energy savings.

The results shown in Figure 5 (right side) 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: OP(d) varies from 155 to 102500 microseconds, and DP varies from 2529 to 49653 microseconds. This points to the fact that our heuristics are extremely powerful in dynamic situations.

Rejection ratio (RR) in this simulations is as follows. From 0 to 30 tasks, rejection ratio is equal to $RR = 0\%$. For 40 tasks $RR = 7.5\%$, which indicates that 3 out of 40 tasks, in average, are rejected from execution. For 50 tasks $RR = 24\%$, for 60 tasks $RR = 35\%$, for 70 tasks, $RR = 42.8\%$, and for 80 tasks, $RR = 47.5\%$.

From the % of energy savings in Figure 5, the confidence intervals computed for 10 to 80 tasks are within the range $[-7\%\ \%ES, +7\%\ \%ES]$. Table 6 shows the results of Figure 5 with 40 Tasks and their corresponding confidence intervals (CI) (assuming a confidence level of 95%).

	SGA	SGA	OP(d)	DP
% ES	21	25	13	27
CI	[20.1,21.8]	[24.1,25.3]	[12.6,13.3]	[25.6,27.9]

Table 6. Confidence Intervals for 40 Tasks.

The results shown in Figure 6 indicate how important is to consider an appropriate number of speed levels for achieving a high percentage of energy savings. As shown in Figure 6, under moderate, *realistic* number of speed levels (between 3 and 30), the EGA algorithm outperforms the OP(d) algorithm. However, for more than 30 speed levels OP(d) algorithm outperforms the EGA algorithm, because the system approaches a continuous voltage setting, in which OP(d) is close to the optimal OP(c).

The run-time computed (shown in Figure 6), indicate that the OP(d) algorithm has very little sensibility to the number of speed levels: the run-time of the OP(d) algorithm varied from 6900 to 7100 microseconds. In contrast, our Greedy Algorithms increased their run-time with higher number of speed levels, but still remained well under OP(d). For this experiments, the run-time of the Greedy Algorithms varied from 99 to 1800 microseconds. The DP algorithm is the most expensive, with one or two orders of magnitude higher run-time than the EGA and SGA algorithms.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

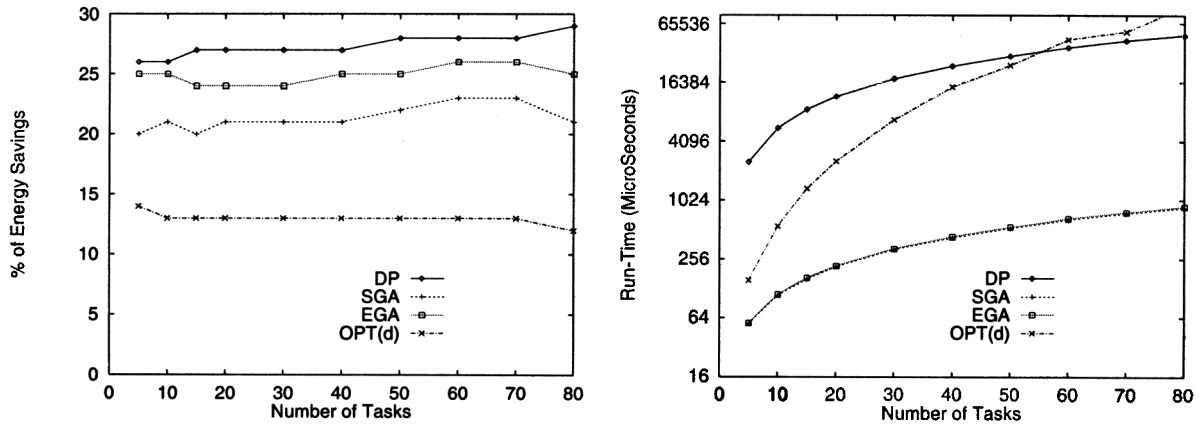

Figure 5. % of Energy Savings and Run-Time (Microseconds)

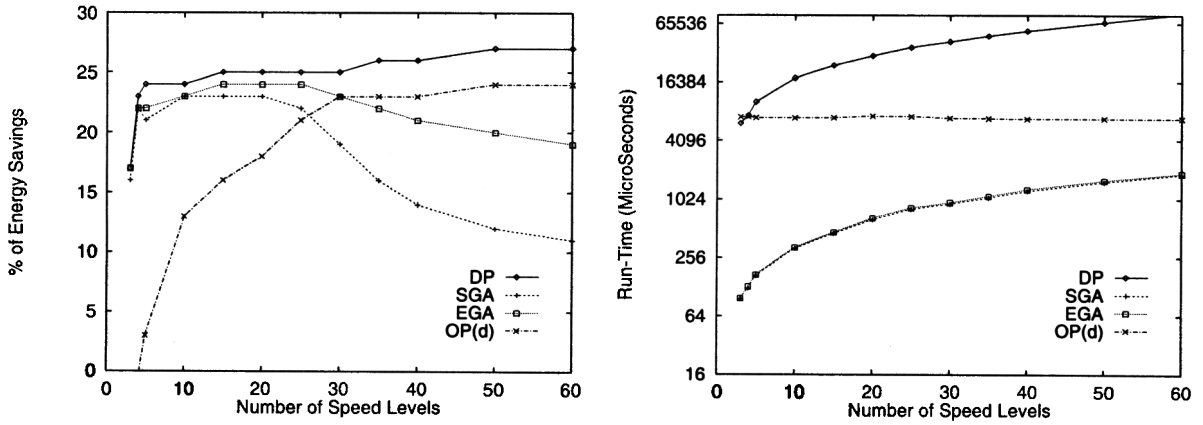

Figure 6. % of Energy Savings and Run-Time (Microseconds)

Rejection ratio for this simulations always yield 0% meaning that no tasks were rejected from execution on any single test.

Further tests were conducted (increasing the number of speed levels) to check when the EGA algorithm and the OP(d) algorithms have similar run-times; this happens when the number of speed levels is approaching 100 (not shown, due to space constraints).

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.

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 are currently extending 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. Mossé, P. Mejia-Alvarez. "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics". *EuroMicro Conference on Real-Time Systems*, June 2001.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 **IEEE**

[2] H. Aydin, R. Melhem, D. Mossé, P. Mejia-Alvarez. "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] G. Buttazzo and F. Sensini. "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments", *IEEE Transactions on Computers*, Vol. 48, No. 10, October 1999.

[5] 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.

[6] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava. "Power Optimization of Variable Voltage Core-Based Systems". In *Design Automation Conference*, 1998.

[7] 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.

[8] 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*, December 1998.

[9] Intel, Microsoft, Compaq, Phoenix and Toshiba. "ACPI Specification", *http://developer.intel.com/technology/IAPC/tech.*

[10] Intel StrongARM SA-1100 microprocessor developer's manual.

[11] 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.

[12] M. Joseph, P. Pandya. "Finding Response Times in a Real-Time System", *Computer Journal*, pp.390-395, Oct. 1986.

[13] 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.

[14] E. Lawler. "Fast Approximation Algorithms for Knapsack Problems". *Mathematics of Operations Research*, Nov. 1979.

[15] C.L. Liu, J. Layland. "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments", *J. ACM*, 20(1). Jan. 1973.

[16] G. Lipari, G. Buttazzo. "Schedulability Analysis of Periodic and Aperiodic Tasks with Resource Constraints", *J. of Systems Architecture*, (46). 2000.

[17] J.R. Lorch, A.J. Smith. "Improving Dynamic Voltage Scaling Algorithms with PACE". In *Proc. of ACM SIGMETRICS Conference* Cambridge, MA, June 2001.

[18] S. Martello and P. Toth. "Knapsack Problems. Algorithms and Computer Implementations". *Wiley*, 1990.

[19] D. Mossé, 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)*, 2000.

[20] A. Dudani, F. Mueller, Y. Zhu. "Energy-Conserving Feedback EDF Scheduling for Embedded Systems with Real-Time Constraints." *ACM SIGPLAN Joint Conference on Languages, Compilers and Tools for Embedded Systems (LCTES'02)*, June 2002.

[21] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems". *In Proceedings of the 18th. ACM Symposium on Operating System Principles (SOSP'01)*, 2001. ACM Press.

[22] D. Pisinger. "A Minimal Algorithm for the Multiple-Choice Knapsack Problem", *European Journal of Operational Research*, 83. 1995.

[23] Y. Shin and K. Choi. "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems". In *Proc. of the Design Automation Conference*. 1999.

[24] D. Shin, W. Kim, J. Jeon, J. Kim and S.L. Min. "SIMDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms". In *Proc. of the Workshop on Power-Aware Computer Systems (PACS'02)*, Feb, 2002

[25] P. Sinha, A. Zoltners. "The Multiple Choice Knapsack Problem". *Operations Research*, May-June 1979.

[26] 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.*

[27] www.transmeta.com

[28] F. Yao, A. Demers, S. Shenker. "A Scheduling Model for Reduced CPU Energy". *IEEE Annual Foundations of Computer Science*, 1995.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 **IEEE**