Resource Allocation for Independent Real-Time Tasks in Heterogeneous Systems for Energy Minimization[*]

YANG YU AND VIKTOR K. PRASANNA
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-2562, U.S.A.
E-mail: {yangyu, prasanna}@halcyon.usc.edu

In recent years, power management and power reduction have become critical issues in portable systems that are designed for real-time use. In this paper, we study the problem of statically allocating a set of independent real-time tasks to a system consisting of heterogeneous processing elements, each enabled with discrete Dynamic Voltage Scaling. The goal is to minimize the overall energy dissipation of the system without violating the real-time requirements of the tasks. The problem is first formulated as an extended Generalized Assignment Problem. A linearization heuristic (LR-heuristic) is then extended to solve the problem. An analysis of the upper bound on the number of tasks that the heuristic may fail to allocate is also presented. Our experiments show that when the average utilization of the system is high, the LR-heuristic achieves 15% off the optimal energy dissipation for small size problems, while the performance of a classic greedy heuristic is around 90% off the optimal. A relative performance improvement of up-to 40% over the classic greedy heuristic is also observed for large size problems. Finally, an analytical performance comparison between the LR-heuristic and the greedy heuristic is presented.

Keywords: energy minimization, real-time, task allocation, generalized assignment problem, linearization heuristic

1. INTRODUCTION

In recent years, power management and power reduction have become increasingly important in portable systems that are designed for real-time use. These systems must be designed to meet both functional and timing requirements. Thus, the quality of service delivered by such systems depends on both the accuracy of computations and their timeliness. The performance as well as the limited energy constraints require implementing different parts of the systems in dedicated hardware blocks. As a result, modern real-time systems [16] are generally composed of a set of heterogeneous processing elements (PEs), where a PE can be a general-purpose processor, a RISC core, or a field-programmable gate array. Such heterogeneous systems may be geographically distributed, or may reside on a single board, yielding heterogeneous multiprocessors that exploit task-level parallelism in applications. Examples of such systems are mobile computing environment [14] and distributed embedded systems [8], among others. Due

Received May 15, 2002; accepted July 25, 2002.
Communicated by Biing-Feng Wang, Stephan Olariu and Gen-Huey Chen.
[*] This work is supported by the DARPA Power Aware Computing and Communication Program under contract No. F33615-C-00-1633. A preliminary version of the paper was presented at the 2002 International Conference on Parallel and Distributed Systems, Chungli, Taiwan.

to the limited energy supply of such systems, hardware components, protocols, and applications should be designed with the goal of minimizing energy dissipation. Furthermore, the capability of reducing the energy dissipation in such systems while meeting the real-time requirements largely depends on the allocation of system resources. Hence, a pre-runtime resource allocation algorithm that takes into consideration the real-time constraints is crucial. However, to determine a “good” resource allocation for energy minimization in such systems is challenging because of the need to address real-time constraints and system heterogeneity. More specifically, the energy dissipation of the system must be carefully balanced against the desired system performance.

Typically, power management and reduction can be achieved by using two methods: (1) activity based dynamic power management [21] and (2) dynamic supply voltage scaling [27]. The first approach brings a processor into a power-down mode, where only certain parts of the computer system (e.g., clock generation and time circuits) are kept running, while the processor is in an idle state. However, the applicability of dynamic power management techniques in real-time systems is limited, due to the latency overhead for state transition. The second approach, *Dynamic Voltage Scaling* (DVS), is based on exploiting the convex relation between the CPU supply voltage and power dissipation. The rationale behind the DVS technique is to stretch out the task execution time through CPU frequency and voltage reduction. Although systems capable of operating on an almost continuous voltage/frequency spectrum are becoming a reality, most of the contemporary processors that support DVS use a few discrete voltage levels. Two example processors that support discrete DVS are (1) the Crusoe processor [29], which can adjust the clock frequency from 200 to 700 MHz and the corresponding supply voltage from 1.1V to 1.6V, in 33 MHz steps, and (2) the ARM7D processor [28], which can run at 33 MHz (5V supply voltage) and 20 MHz (3.3V supply voltage).

In this paper, we study the problem of statically allocating a set of independent real-time tasks onto a system consisting of heterogeneous processing elements, each equipped with the discrete DVS feature. The tasks considered in this paper are assumed to be periodic. Sporadic and aperiodic tasks can be treated as periodic by allotting to them a periodically-replenished execution budget [1]. These problems require determining the assignment of tasks onto processors as well as the voltage setting of each task. In general, such allocation problems are NP-complete. Therefore, heuristics are desired to obtain sub-optimal solutions. The allocation problem is first formulated as a Integer Linear Programming (ILP) problem, which can be viewed as an extension of the traditional Generalized Assignment Problem [25]. An extended LR-heuristic [25] is then used to solve the problem. We present a lower bound on the number of tasks that the LR-heuristic may fail to allocate. Our experiments show that when the real-time constraints are tight, the LR-heuristic achieves 15% off the optimal energy dissipation for small size problems, while the performance of a classic greedy heuristic is around 90% off the optimal. A relative performance improvement of up to 40% over the classic greedy heuristic is also observed for large size problems. Finally, we present an analytical performance comparison of the LR-heuristic and the greedy heuristic.

The rest of the paper is organized as follows. A brief discussion of related work is presented in section 2. The system and application models are discussed in section 3. A formal ILP formulation of the problem based on the models is presented in section 4. The LR-heuristic for solving the problem is presented and analyzed in section 5. Experimental results are presented in section 6. Concluding remarks and a discussion of future work are given in section 7.

2. RELATED WORK

Studies on scheduling strategies for have aimed to adjust the CPU speed so as to reduce energy dissipation in the context of a non-real-time environment. One approach was proposed in [27], where time is divided into 10-50 msec intervals, and the CPU speed is adjusted using a task-level scheduler based on processor utilization over the preceding interval. A comparison of several predictive and non-predictive approaches to voltage changes was presented in [9]. It was concluded in [9] that smoothing helps more than the prediction. A stochastic model for prediction of execution times for streaming multimedia applications on a frame-by-frame basis was developed in [23]. An integrated DVS and DPM approach was then proposed to save energy based on the stochastic model. In [24], a workload prediction strategy based on adaptive filtering of the past workload profile was proposed, and several filtering schemes were analyzed.

Most of the existing related works are limited to using voltage scaling for energy minimization on single-processor systems. Some examples are the works in [3, 4, 10-12, 20, 22, 30]. One of the earliest work, [30], provided an optimal static scheduling algorithm to minimize the total energy dissipation while satisfying the relative deadline of all tasks. Synthesis techniques for core-based real-time system-on-chip were studied in [10, 11]. A non-preemptive variable voltage scheduling heuristic with the assumption of zero delay in changing voltage levels was developed in [10]. In [11], preemptive variable voltage scheduling that takes into account the inherent limitation on the voltage changing rates was considered. Based on the assumption that the voltage cannot change continuously, a static voltage scheduling problem was studied in [12] and formulated as a ILP problem. An efficient solution for scheduling periodic real-time tasks with (potentially) different power consumption characteristics was presented in [3]. The equivalence of the static scheduling problem with the reward-based scheduling problem with concave reward functions was shown in [4]. In [20], a class of algorithms was proposed to modify the OS's real-time scheduler and task management service, thus providing energy savings while preserving deadline guarantees. In [22], an off-line fixed-priority scheduling technique was presented. The above work established basic theories for using voltage scaling. However, the problem becomes significantly different in the environment of networked embedded systems, where multiple processors are available.

The studies most relevant to our problem include [7, 13, 17, 32, 33]. An energy minimization technique for independent periodic tasks on homogeneous multiprocessor platforms was discussed in [7]. It was assumed in [7] that an EDF scheduling policy and task migration (without penalty) were available in the system. The problem was formulated as an optimization problem with a quadratic objective function and non-linear constraint functions.

The technique proposed in [13] addresses the problem of synthesizing a set of independent tasks on a multi-processing system with the continuous DVS feature and non-preemptive scheduling policy. In that work, the problem of allocating the tasks was solved by first assigning tasks to the processors and then adjusting the voltage levels of the processors. The task assignment procedure iteratively selects a task for assignment based on a parameterized objective function and then determines the suitable processor for executing the task using another parameterized objective function. A meta-algorithm was also developed to statistically determine the parameters involved in the two objective functions.

A power-conscious joint scheduling technique for periodic task graphs and aperiodic tasks in distributed real-time systems was presented in [17]. That work assumed a non-preemptive task scheduling approach and considered the communication cost between tasks. Resource allocation was carried out in two steps. A feasible static resource allocation, task assignment and scheduling was first obtained using a system synthesis tool [6] along with a genetic algorithm. The next step is re-scheduling of tasks on each individual node that tries to evenly distribute the slackness among tasks. A runtime mechanism is then used to scale the voltage.

For frame-based tasks and homogeneous multi-processor environments, a dynamic processor supply voltage adjustment mechanism using slack reclamation was discussed in [33]. A longest-task-first partitioning heuristic is employed to assign tasks onto processors. A slack sharing algorithm is then used at runtime to determine how to adjust the voltage levels of the processors.

In [32], a two-phase framework was presented to investigate the problem in which precedence constraints are present between tasks, where identical periods for all tasks and a non-preemptive scheduling are assumed. The first phase determines appropriate assignment of tasks to processors using a greedy heuristic. The second phase determines the voltage levels for executing tasks using a convex programming mechanism.

The contribution of our work is to systematically formulate the resource allocation problem in real-time systems as an extended Generalized Assignment Problem (GAP). Because of the inherent similarity between these two problems, techniques that are available for solving GAP can be extended to solve the resource allocation problem. In this paper, the extension of a linearization heuristic to solve GAP is studied. Most relevant techniques tend to consider the assignment of tasks and the settings of voltage levels as two separate subproblems and solve them in two consecutive steps. However, by simultaneously solving the two aforementioned subproblems in a single formulation, we can efficiently explore the interrelationship between the two subproblems.

3. PROBLEM DEFINITION

System Model: The system consists of a set of m PEs, $\{PE_1, PE_2, \ldots, PE_m\}$. Each PE is equipped with discrete DVS feature and can adjust its voltage independently of others. Let V_k denote the number of discrete voltage levels of PE_k, $k = 1, 2, \ldots, m$. In addition, an Earliest Deadline First (EDF) [15] scheduling policy is assumed to be employed by each PE.

Application Model: A set of n independent periodic real-time tasks, $\{T_1, T_2, \ldots, T_n\}$, are considered. The period of T_i is denoted by P_i, which is assumed to be equal to the relative deadline of each instance of T_i [15]. This means that each instance of a task must complete execution before the next instance of the task is activated. The *planning cycle* of the system is defined as the least common multiple of the periods of all tasks, denoted as

LCM. It is well known that for the above real-time application, it suffices to analyze the behavior of the system within a planning cycle.

The workload of a task is measured based on the worst-case number of CPU cycles required to execute the task, which can be different on different PEs, due to system heterogeneity. The worst-case number of CPU cycles required by T_i to execute on PE_k is assumed to be a finite positive number, denoted by C_{ik}. The execution time of T_i on PE_k under a constant speed S (given in cycles per second) of PE_k is $t_{ik}(S) = \frac{C_{ik}}{S}$. The value of S is determined by the supply voltage of PE_k, which can be set to V_k discrete levels. In addition, the utilization of T_i on PE_k under speed S, $u_{ik}(S)$, is defined as the ratio of the execution time to the period of the task. Thus, we have $u_{ik}(S) = \frac{t_{ik}(S)}{P_i}$. Let $SMAX_{ik}$ denote the maximum speed of executing T_i on PE_k. In a heterogeneous environment, the value of $SMAX_{ik}$ for a particular T_i can be different for different PEs. However, the effect of such a difference can be captured by normalizing C_{ik} with $SMAX_{ik}$, while setting the value of $SMAX_{ik}$ to 1. Thus, without loss of generality, we assume that $SMAX_{ik}$ equals 1 for all T_i and PE_k.

In each PE, the voltage is assumed to dynamically switch, if necessary, upon the arrival or the resumption of execution (because of preemptive task scheduling) of task instances. An upper bound on the number of such switches during the execution of an instance of T_i is given by $\sum_{j=1}^{n} \left\lceil \frac{P_i}{P_j} \right\rceil$. The time overhead associated with this switching is assumed to be included in the worst-case workload of the corresponding task. The power consumption of task T_i on PE_k under speed S is denoted by $g_{ik}(S)$, a strictly increasing convex function, represented by a polynomial of at least second degree [11]. By assuming that the value of S is fixed during the time used to execute T_i, we can calculate the corresponding energy dissipation as $g_{ik}(S)t_{ik}(S)$ (recall that $t_{ik}(S)$ is the corresponding execution time). Due to system heterogeneity, the exact form of the polynomial function, g_{ik}, can be different for executing different tasks on the same PE and/or for executing the same task on different PEs.

Resource Allocation: A resource allocation is defined as an assignment of all the tasks to the PEs, together with the setting of the voltage level for each task on the corresponding PE. Each task can be assigned to exactly one PE and can be executed with a fixed voltage level on that PE.

Assuming that a set of tasks, T, is allocated on PE_k, the EDF schedule of tasks in T is *feasible* if and only if the total utilization of all tasks in T does not exceed the computation capacity of PE_k [15], i.e., $\sum_{T_i \in T} u_{ik}(S_{ik}) \le 1$, where S_{ik} is the speed of executing T_i on PE_k. An allocation is called *feasible* if for every PE_k in the system, the EDF schedule of tasks allocated on PE_k is feasible. A feasible allocation is optimal if the overall energy dissipation of the system is minimal among all feasible allocations. Because different tasks may have different periods, the overall energy dissipation is calculated as the energy dissipation of the system during a planning cycle.

4. INTEGER LINEAR PROGRAMMING FORMULATION

The problem defined in section 3 essentially requires an assignment of tasks to the voltage levels that are available in the system. Let $\hat{m}$ denote the total number of voltage levels in the system, i.e., $\hat{m} = \sum_{k=1}^{m} V_k$. Also, we label the d-th voltage level of PE_k as the j-the voltage level of the system, denoted by VL_j, where $j = \sum_{i=1}^{k-1} V_i + d$. Thus, by referring to a voltage level, we unambiguously mean the corresponding PE and the corresponding voltage level of the PE. Let $VLG(k)$ denote the set of voltage levels of PE_k.

Let u'_{ij} denote the utilization of task T_i when executed on voltage level VL_j. Similarly, let e_{ij} denote the energy dissipation for executing an instance of T_i on VL_j. Since we are optimizing the system energy dissipation during a planning cycle, let e'_{ij} denote the total energy dissipation for executing T_i on VL_j during a planning cycle. Thus, we have $e'_{ij} = e_{ij} \frac{LCM}{P_i}$. The values of u'_{ij}'s and e'_{ij}'s can be calculated based on the analysis given in section 3. Given an allocation, the utilization of voltage level VL_j, U_j, is defined as the sum of the utilization of tasks that are assigned to VL_j. Therefore, for any feasible allocation, we must have $\sum_{VL_j \in VLG(k)} U_j \le 1, k = 1, \ldots, m.$

Let $\{x_{ij}\}$ be a set of 0-1 variables such that x_{ij} equals one if T_i is assigned to VL_j, and zero otherwise. The problem can now be formulated as the following ILP problem, ILP(1):

$$
\begin{array}{lll}
\text{Minimize} & \sum_{i=1}^{n} \sum_{j=1}^{\hat{m}} e'_{ij} x_{ij} & \\
\text{Subject to} & \sum_{i=1}^{n} u'_{ij} x_{ij} - U_j \le 0 & j = 1, 2, \ldots, \hat{m}, \quad (1) \\
 & \sum_{VL_j \in VLG(k)} U_j \le 1 & k = 1, 2, \ldots, m, \quad (2) \\
 & \sum_{j=1}^{\hat{m}} x_{ij} = 1 & i = 1, 2, \ldots, n, \quad (3) \\
 & x_{ij} \in \{0, 1\} & i = 1, 2, \ldots, n, \\
 & & j = 1, 2, \ldots, \hat{m}. \quad (4)
\end{array}
$$

This formulation is in the form of a Generalized Assignment Problem (GAP) [25] except for constraints (2). More specifically, the capacity of resources, in terms of the upper-bound on the utilization of voltage levels, is defined in groups in ILP(1), whereas in the case of GAP, they are defined individually. If the value of U_j's in ILP(1) are known, a corresponding GAP formulation can be obtained by substituting the U_j's with their corresponding values and removing constraints (2). Intuitively, the values of U_j's can be determined by solving the linear relaxation of ILP(1) obtained by replacing constraints (4) with non-negativity constraints. Let LP(1) denote the linear relaxation of ILP(1).

5. RELAXATION HEURISTIC

In this section, we first give an upper bound on the number of split tasks (to be defined later) in a basic solution [19] of LP(1). A linear relaxation heuristic proposed in [25] for solving GAP is then extended to solve ILP(1). Finally, an upper bound on the number of tasks that the heuristic may fail to allocate is derived.

For any solution to LP(1), a task T_i is said to be a *split task* if there exist j and j', such that $j \neq j'$ and x_{ij}, $x_{ij'} > 0$. Task T_i is said to be *integrally assigned*, otherwise. Also, a PE or voltage level is said to be allocated to capacity if its utilization equals 1 according to the (partial) allocation determined by a solution of LP(1). Due to the similarity between ILP(1) and GAP, it can be easily shown that the upper-bound on the number of split jobs in a basic solution to LP(1) is the number of voltage levels allocated to capacity (Theorem 1 in [25]), which is $\hat{m}$. However, because of the special properties introduced by constraints (2) in ILP(1), we show an alternative formulation of the problem, and, consequently, improve the upper-bound to m.

Lemma 5.1 In every basic solution of LP(1), the number of split tasks is at most equal to the number of PEs allocated to capacity.

Proof: Observing that in LP(1), the sum of the utilization of voltage levels that belong to any $VLG(k)$ cannot exceed 1, we can obtain the following alternative linear programming formulation, LP(2):

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{n}\sum_{j=1}^{\hat{m}} e'_{ij}x_{ij} & \\ \text{Subject to} & \sum_{i=1}^{n}\sum_{VL_j \in VLG(k)} u'_{ij}x_{ij} \leq 1 & k = 1, 2, \ldots, m, \\ & \sum_{j=1}^{\hat{m}} x_{ij} = 1 & i = 1, 2, \ldots, n, \\ & x_{ij} \geq 0 & i = 1, 2, \ldots, n, \\ & & j = 1, 2, \ldots, \hat{m}. \end{array}$$

It is easy to verify that the set of x_{ij}'s in any basic solution to LP(1) constitutes a basic solution to LP(2). Also, from any basic solution to LP(2), the values of U_j's can be determined and, consequently, form a basic solution to LP(1).

Given a basic solution to LP(2), the utilization of T_i on PE_k can be calculated as $u_{ik} = \sum_{VL_j \in VLG(k)} u'_{ij}x_{ij}$. Let u_{ik}^{max} denote the maximal value among u'_{ij}'s, where $VL_j \in VLG(k)$. Similarly, let u_{ik}^{min} denote the minimal value among u'_{ij}'s. As an intermediate value between u_{ik}^{max} and u_{ik}^{min}, u_{ik} can be represented as $u_{ik}^{max}y_{ik} + u_{ik}^{min}y'_{ik}$, where y_{ij}, $y'_{ik} \geq 0$ and $y_{ik} + y'_{ik} = \sum_{VL_j \in VLG(k)} x_{ij}$. More specifically, we have $y_{ij} = \frac{\alpha u_{ik}^{max} - u_{ik}}{u_{ik}^{max} - u_{ik}^{min}}$ and $y'_{ij} = \frac{u_{ik} - \alpha u_{ik}^{min}}{u_{ik}^{max} - u_{ik}^{min}}$, where $\alpha = \sum_{VL_j \in VLG(k)} x_{ij}$. We can now consider the following linear programming formulation, LP(3), for which a feasible solution is desired:

$$\begin{array}{lll} \text{Subject to} & \sum_{i=1}^{n} u_{ik}^{\max}y_{ik} + u_{ik}^{\min}y'_{ik} \leq 1 & k = 1, 2, \ldots, m, \\ & \sum_{k=1}^{m} (y_{ik} + y'_{ik}) = 1 & i = 1, 2, \ldots, n, \\ & y_{ik}, y'_{ik} \geq 0 & i = 1, 2, \ldots, n, \\ & & k = 1, 2, \ldots, m. \end{array}$$

Clearly, for any basic solution to LP(2), there is a corresponding basic solution to LP(3). Furthermore, LP(3) forms a linear relaxation of the allocation problem with multiple variable-speed PEs [26]. Any basic solution to LP(3) is known to have the property that the number of split tasks is at most equal to the number of PEs allocated to capacity [26]; thus, the claim follows. ❑

Let $U_j^{\max}$ denote the maximum possible value of U_j (recall that U_j is the utilization of voltage level VL_j). Given a partial assignment of tasks, it is easy to see that the value of $U_j^{\max}$ equals the remaining capacity of the PE that VL_j belongs to. The remaining capacity of a PE can be calculated by subtracting from 1 the sum of the utilization of tasks that are integrally assigned to the PE, according to the partial assignment. Specifically, assuming that VL_j belongs to PE_k, we have $U_j^{max} = 1 - \sum_{VL_l \in VLG(k)} \sum_{x_{il}=1} u'_{il}$. In addition, a variable x_{ij} is defined as *useless* if $u'_{ij} > U_j^{max}$. Here, the definition of a useless variable is an extension of its original definition in [25], due to constraints(2) in ILP(1). Now, the linear relaxation heuristic, LR-heuristic, proposed in [25] can be extended to solve ILP(1). The LR-heuristic is as follows:

Input: an instance of ILP(1)
Output: an assignment of tasks to voltage levels

```
Step 0) Remove all useless variables; if no variables remain, stop.
Step 1) Solve the linear relaxation.
Step 2) Fix all $x_{ij}$'s of value 1; delete the corresponding tasks and update the remaining capacity of PEs.
Step 3) Go to Step 0.
```

Initially, the values of U_j^{max}'s are set to 1, for j = 1, 2, …, $\hat{m}$, since no task has yet been assigned. Once a (partial) assignment is obtained by executing Step 1 at least once, the values of U_j^{max}'s are updated accordingly. The main idea of LR-heuristic is to delete useless variables after fixing all x_{ij}'s of value 1 in Step 2. It is known (Theorem 2 in [25]) that there is at least one useless variable if any split task exists in a basic solution of the linear relaxation.

Since the number of split tasks in any basic solution is at most m, it is easy to check that Step 1 of LR-heuristic is executed at most m + 1 times. It can be further shown that the number of tasks the LR-heuristic fails to allocate is bounded.

Corollary 5.1 If ILP(1) has a feasible solution, then the LR-heuristic fails to allocate at most m – 1 tasks.

The proof follows from Lemma 5.1 and Theorem 4 in [26].

6. EXPERIMENTAL RESULTS

In this section, we show the quality of solutions produced by LR-heuristic obtained in our experiments. A simulator based on the system and application models presented in section 3 was developed to evaluate the performance of LR-heuristic in solving our problem using synthetic task sets. The goals of our experiments were: (1) to measure and compare the performance of LR-heuristic against the optimal solution and a classic greedy heuristic (to be explained later), and (2) to measure the impact of the variation of several system parameters (e.g., the utilization of the system and the number of voltage levels per PE) on the performance of the LR-heuristic.

6.1 Experimental Procedure

The experiments were divided into two sets, for small size problems and large size problems, respectively. We explain the parameters settings for both sets in this section.

For small size problems, the number of PEs was fixed at 5 (so that the optimal solution could be computed in a reasonable amount of time by using LINDO), while the number of tasks varied from 20 to 40. The system heterogeneity was captured by the distribution of the worst-case number of CPU cycles (C_{ik}) required by different tasks on different PEs. It could be characterized by means of task and PE heterogeneities. To emphasize the impact on system performance due to high heterogeneity, the results for high task and PE heterogeneities are given in this paper. Let $\mathcal{C}$ denote the matrix composed by $\{C_{ik}\}$, where $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$. The $\mathcal{C}$ matrix was generated using a Gamma distribution based method [2]. The mean value along the task axis, μ_{task}, was set to 200. Two other parameters that indicate the task and PE heterogeneities, V_{task} and V_{PE}, were both set to 0.5.

P_i, the period of task T_i, was generated based on the $\mathcal{C}$ matrix. Let w_i be the largest value among the elements in the i-th row of $\mathcal{C}$, and let $X = \frac{n}{m}$ (recall that n is the number of tasks, and that m is the number of PEs). P_i is calculated as $\frac{w_i}{U_{sys}} \cdot X$, where U_{sys} is a pre-specified positive value that approximates the average utilization of the entire system when assuming that all the processors are running at their maximum speeds. A large value of U_{sys} indicates a high level of contention over resources caused by tasks. Note that from the above equation, the utilization of an individual task decreases when the number of tasks increases with a fixed number of PEs, so that the utilization of the entire system is sustained. To set the value of *LCM* within some reasonable range, the value of P_i was adjusted to some close value, so the value of *LCM* was at most at 36000.

For small size problems, the number of voltage levels for all PEs was fixed at 4. The maximum CPU speed of each PE was set to 1.0, and the minimum speed was set to 0.25. It was assumed that other CPU speeds were distributed uniformly between the maximum and minimum speeds. Therefore, the other two levels of CPU speed were set to 0.75 and 0.5. The energy function of T_i on PE_k, $g_{ik}(S)$, was of the form $a_{ik} \ldots S^{b_{ik}}$, where S is the CPU speed of PE_k, and a_{ik} and b_{ik} were random variables with uniform distributions between 2 and 10, and 2 and 3, respectively [18].

For large size problems, the number of PEs was fixed at 10, while the number of tasks varied from 60 to 100. The number of DVS levels per PE was set to 8. Other parameters were the same as those for small size problems.

The greedy heuristic (referred to as Greedy hereafter) is an extension of the min-min heuristic that is widely used for task allocation in heterogeneous computing [5]. The original objective function of the min-min heuristic is to minimize the *makespan* [5] of a set of tasks. Here, the heuristic is modified so that the overall energy dissipation of the system is minimized, while satisfying the utilization constraint of each PE. The pseudo code for Greedy is shown in Fig. 1.

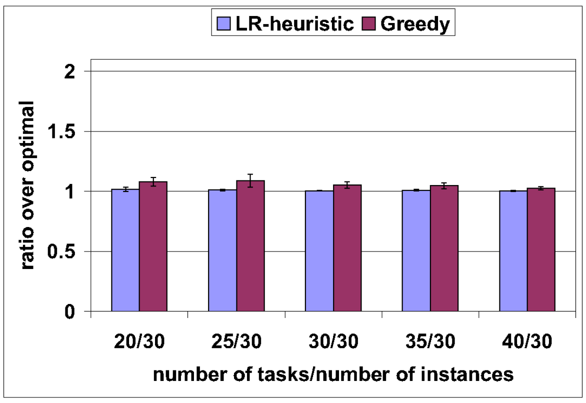
```
Begin
1.  $T^* \leftarrow T$;
2.  $U_j^{\max} \leftarrow 1$ for $j = 1, 2, \ldots, \hat{m}$
3.  While $T^*$ is not empty, Do
4.      For each $T_i \in T^*$, find the voltage level, $VL_j$, such that $e'_{ij}$ is minimal among all
        the voltage levels and $u'_{ij} \leq U_j^{\max}$. Record the information as a turple ($T_i$, $VL_j$, $e'_{ij}$)
5.      Select the tuple that gives the minimal value of $e'_{ij}$
6.      Allocate the task in the selected tuple to the corresponding voltage level in the
        turple and remove the task from $T^*$
7.  Update $U_j^{\max}$ of all voltage levels
End
```

Fig. 1. Pseudo code for the Greedy heuristic.

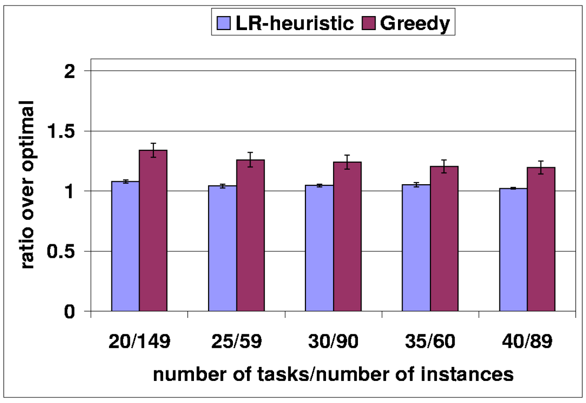
6.2 Results

The experimental results for small size problems when U_{sys} was set to 40%, 50%, and 67% are shown in Fig. 2. It shows the ratio of the overall system energy dissipation obtained from two heuristics to the optimal. Each bar represents the average value of the ratio over a specific number of instances when both heuristics succeeded in finding a feasible mapping. The number of instances for each case (shown next to the number of tasks in the figure) was chosen to be large enough such that the presented ratio had a 95% confidence interval with 5% (or better) precision. The number of instances could differ for different cases. The confidence intervals are indicated by the short lines at the top of each bar. During the experiments, if any heuristic failed in any instance, that instance was excluded from the calculations of the average ratio and the confidence interval. However, these instances were included in the calculation of the miss rate of each heuristic (to be explained later). The plots clearly show that LR-heuristic always outperformed Greedy. When U_{sys} = 40%, both heuristics performed quite well. When the value of U_{sys} increased (i.e., the real-time constraints became tighter, and the level of contention over resources became higher), the performance of both heuristics became worse. However, the performance of LR-heuristic was still quite acceptable when U_{sys} = 67%, achieving 15% off the optimal, while the performance of Greedy was around 90% off the optimal.

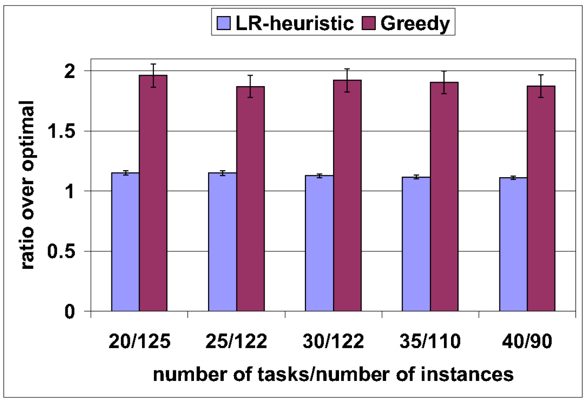
It was observed that when U_{sys} increased, the number of instances for which Greedy failed to find a feasible allocation increased rapidly. On the contrary, the LR-heuristic still succeeded in all instances, even when U_{sys} = 67%. Table 1 shows the miss rate (the number of instances that a heuristic failed, normalized with respect to the total number of instances) and the average number of unallocated tasks over all instances of both heuristics when U_{sys} = 67% and 50%. The astonishingly high miss rate of Greedy when U_{sys} = 67% indicates the inappropriateness of using Greedy for problems with high resource contention.



(a) U_{sys} = 40%.



(b) U_{sys} = 50%.



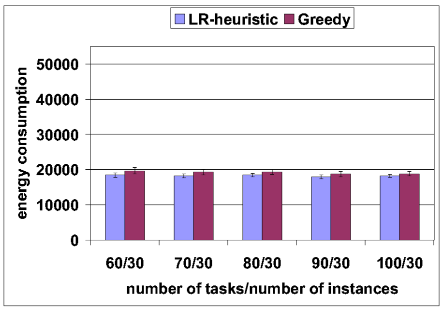
(c) U_{sys} = 67%.

Fig. 2. Experimental results for small size problems.

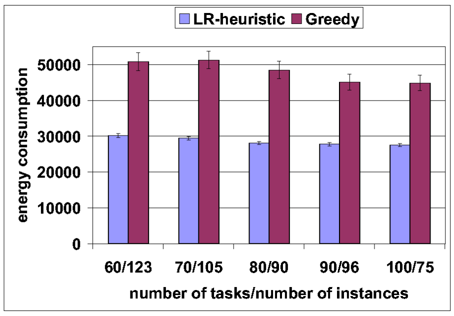
Table 1. The miss rate and the average number of unallocated tasks of LR-heuristic and Greedy for small size problems.

U_{sys}		50%					67%				
# of tasks		20	25	30	35	40	20	25	30	35	40
miss rate (%)	LR-heuristic	0	0	0	0	0	0	0	0	0	0
	Greedy	1	2	0	0	1	40	49	59	72	73
unallocated tasks	LR-heuristic	0	0	0	0	0	0	0	0	0	0
	Greedy	0.01	0.03	0	0	0.02	0.69	1.04	1.48	2.21	2.34

Fig. 3 shows the relative performance of LR-heuristic and Greedy for large size problems when U_{sys} = 60% and 90%. Again, the number of instances for each case was chosen to be large enough such that the presented data had a 95% confidence interval with 5% (or better) precision. Similar trends in performance were observed when the value of U_{sys} increased. When U_{sys} equaled 90%, the LR-heuristic showed upto 40% improvement over the performance of Greedy. It was noticed that for the same value of U_{sys}, the relative performance of Greedy was better for large size problems, compared with small size problems. This may be due to the fact that for the same value of U_{sys}, the deadline constraints for large size problems are actually not as tight as those in the case of small size problems because (1) there are 8 DVS levels for each PE in large size problems, whereas there are 4 DVS levels in small size problems, and (2) for large size problems, the size of $\mathcal{C}$ matrices increases, hence leading to a higher likelihood of getting larger values of w_i's (recall that w_i is the largest value within the i-th row of $\mathcal{C}$), and consequently, larger periods for tasks.



(a) U_{sys} = 60%



(b) U_{sys} = 90%

Fig. 3. Experimental results for larger size problems.

We also conducted experiments to study the impact of the number of DVS levels on the performance of LR-heuristic and Greedy. The results are shown in Fig. 4. The performance of both heuristics improved when the number of DVS levels increased. Additionally, LR-heuristic was much less sensitive to variation in the number of DVS levels.

Table 2 shows the average running time of both heuristics for small size and large size problems. The running time of LINDO for small size problems is also presented for comparison. The figure shows that when the number of tasks increased, the ratio of the

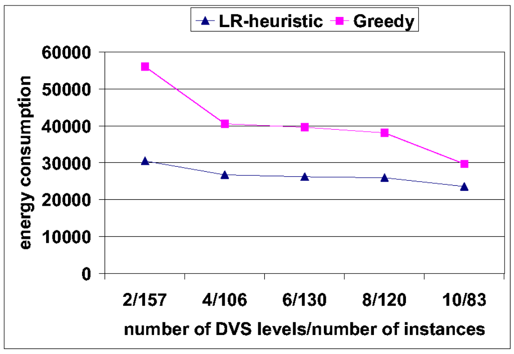


Fig. 4. Performance of heuristics as a function of the number of DVS levels (10 PEs, 80 tasks, U_{sys} = 80%).

Table 2. Average running time (in seconds) of LINDO and both heuristics.

Problem size	5 PEs, U_{sys} = 50%					10 PEs, U_{sys} = 60%				
# of tasks	20	25	30	35	40	60	70	80	90	100
LINDO	0.28	0.26	0.36	0.51	0.37	-	-	-	-	-
LR-heuristic	0.071	0.071	0.078	0.077	0.078	0.26	0.30	0.34	0.37	0.40
Greedy	0.0004	0.0007	0.0009	0.0013	0.0016	0.007	0.009	0.012	0.015	0.02

running time of LR-heuristic to that of Greedy decreased from 160 to 48 in the case of small size problems, and from 37 to 21 in the case of large size problems. Since our problem is an off-line design phase problem, where the running time is not a critical concern, LR-heuristic is a good choice for finding a solution.

6.3 Performance Analysis

An analysis of the performance of the proposed LR-heuristic and the Greedy heuristic (given in section 6.1) is presented in this section.

Lemma 6.1 There exists an infinite number of problem instances in which LR-heuristic can find the optimal solution, while Greedy fails to find a feasible solution.

Proof: Consider a problem instance with $m = n = a$, where a is an integer > 1. Also, let each PE have exactly one voltage level (with the corresponding CPU speed set to 1). Thus, we have $\hat{m} = a$. For each T_i, $1 \leq i \leq a$, the value of P_i is set to some positive constant b. Let $\mathcal{E}$ denote the matrix $\{e_{ij}\}$, where $i, j = 1, 2, \ldots, a$. The $\mathcal{C}$ and $\mathcal{E}$ matrices are illustrated in Fig. 5. In Fig. 5, e is some positive constant. Since the CPU speed of every PE is set to 1, each entry in $\mathcal{C}$, C_{ij}, equals the time needed to execute task T_i on PE_j. It can be verified that for this problem instance, a basic solution of LP(1) is to assign T_i to PE_i, for $i = 1, 2, \ldots, a$. In other words, all tasks are integrally assigned in this basic solution. Therefore, LR-heuristic is able to find a feasible solution of the problem after the first iteration of the heuristic. It is easy to verify that this feasible solution is also the optimal solution of the problem, which dissipates $a \cdot e$ amount of energy in each planning cycle.

If the Greedy heuristic is used to solve the problem, it will first allocate T_1 to PE_a, then T_2 to $PE_{a\text{-}1}$, T_3 to $PE_{a\text{-}2}$, and so on. However, when T_a is the only remaining task to be allocated, no PE can be used to generate a feasible assignment. ❑

	PE_1	PE_2	....	PE_{a-1}	PE_a
T_1	b	b	....	b	b
T_2	b	b	....	b	b
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_{a-1}	b	b	....	b	b
T_a	$b+1$	b	....	b	b

(a) $\mathcal{C}$ matrix

	PE_1	PE_2	....	PE_{a-1}	PE_a
T_1	e	e	....	e	$e-1$
T_2	e	e	....	$e-1$	e
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_{a-1}	e	$e-1$	....	e	e
T_a	$e-1$	e	....	e	e

(b) $\mathcal{F}$ matrix

Fig. 5. The $\mathcal{C}$ and $\mathcal{F}$ matrices for the problem instance in the proof of Lemma 6.1.

Lemma 6.2 There exists an infinite number of problem instances in which the performance (in terms of the overall energy dissipation of the system in a planning cycle) of the solution found by LR-heuristic can be arbitrarily better than the performance of the solution found by Greedy.

Proof: The proof is a slight variation of the proof of Lemma 6.1. We use a similar problem instance except that we change the value of C_{a1} to b and the value of E_{a1} to θ, where $\theta > e + a - 1$. By using an analysis similar to the analysis in Lemma 6.1, we can see that LR-heuristic can find a feasible solution that dissipates $a \cdot e$ amount of energy in each planning cycle, by allocating T_i to PE_i. However, if the Greedy heuristic is used to solve the problem, it will find a solution that dissipates $(e - 1)(a - 1) + \theta$ energy during a planning cycle by allocating T_i to PE_{a-i+1}. The ratio of $(e - 1)(a - 1) + \theta$ over $a \cdot e$ can be made arbitrarily large by varying θ. ❑

7. CONCLUDING REMARKS

This paper has discussed the problem of allocating a set of independent real-time tasks in a heterogeneous system. The problem has been formulated as an extended Generalized Assignment Problem and solved through an extension of LR-heuristic. An upper-bound on the number of tasks that LR-heuristic may fail to allocate has been presented. An analytical comparison of the performance of LR-heuristic and a classic greedy heuristic has also been given.

In the future, we plan to study some related problems that consider: (1) application specified as a set of pipelines or directed acyclic graphs, and (2) continuous DVS features. In the first class of problems, dependency constraints between tasks must be considered. Also, modeling and optimizing the communication time and energy costs offer new challenges, especially in networked sensor environments, where communication is carried out in an ad hoc fashion. Convex optimization techniques may be used to solve the second class of problems. Also, we are interested in designing on-line energy-aware scheduling policies for distributed real-time systems based on the techniques developed in this paper.

This work is a part of the Power Aware Computing/Communication for Mobile Ad Hoc and Sensor Networks (PACMAN) project at USC. Related results can be found at http://pacman.usc.edu.

REFERENCES

1. T. F. Adbelzaher and K. G. Shin, “Period-based load partitioning and assignment for large real-time applications,” *IEEE Transactions on Computers*, Vol. 49, 2000, pp. 81-87.
2. S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, “Task execution time modeling for heterogeneous computing systems,” *9th Heterogeneous Computing Workshop*, 2000, pp. 185-199.
3. H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, “Determining optimal processor speeds for periodic real-time tasks with different power characteristics,” *13th Euromicro Conference on Real-Time Systems*, 2001, pp. 225-232.
4. H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, “Dynamic and aggressive scheduling techniques for power-aware real-time systems,” *22nd Real-Time Systems Symposium*, 2001.
5. T. D. Braun, S. Ali, H. J. Siegel, and N. Beck, “A comparison of eleven static heuristics for mapping a class of independent tasks noto heterogeneous distributed computing systems,” *Journal of Parallel and Distributed Computing*, 2001, pp. 810-837.
6. R. P. Dick and N. K. Jha, “MOCSYN: Multiobjective core-based single-chip system synthesis,” *Design Automation & Test in Europe Conference*, 1999, pp. 263-270.
7. S. Funk, J. Goossens, and S. Baruah, “Energy-minimization techniques for real-time scheduling on multiprocessor platforms,” Technical Report UNC-CS TR01-030m Dept. of Computer Science, University of North Carolina at Chapel Hill, 2001.
8. D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, Prentice-Hall, NJ, 1994.
9. K. Govil, E. Chan, and H. Wasserman, “Comparing algorithms for dynamic speed-setting of a low-power CPU,” *ACM International Conference on Mobile Computing and Networking*, 1995, pp. 13-25.
10. I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava, “Power optimization of variable voltage core-based systems,” *35th Design Automation Conference*, 1998, pp. 176-181.
11. I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, “Synthesis techniques for low-power hard real-time systems on variable voltage processors,” *19th IEEE Real-Time Systems Symposium*, 1998, pp. 178-187.
12. T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” *International Symposium on Low Power Electronics and Design*, 1998, pp. 197-202.

13. D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," *34th Design Automation Conference*, 1997, pp. 697-702.
14. M. J. Kumar and S. Venkatesh, "Power saving schemes for mobile computing environment," *4th International Conference on Advanced Computing*, 1996, pp. 296-302.
15. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real-time environment," *Journal of ACM*, Vol. 20, pp. 46-61.
16. J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, NJ, 2000.
17. J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," *Computer-Aided Design*, 2000, pp. 357-364.
18. P. Mejía-Alvarez, E. Levner, and D. Mossé, "An integrated heuristic approach to power-aware real-time scheduling," *Workshop on Power-Aware Computer Systems*, 2002.
19. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, NJ, 1982.
20. P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *18th ACM Symposium on Operating Systems Principles*, 2001.
21. Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," *36th Design Automation Conference*, 1999, pp. 555-561.
22. G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on voltage variable processors," *38th Design Automation Conference*, 2001, pp. 828-833.
23. T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli, "Dynamic voltage scaling and power management for portable systems," *38th Design Automation Conference*, 2001, pp. 524-529.
24. A. Sinha and A. Chandrakasan, "Dynamic power management in wireless sensor networks," *IEEE Design and Test of Computers*, Vol. 18, 2001, pp. 62-74.
25. M. A. Trick, "A linear relaxation heuristic for the generalized assignment problem," *Naval Research Logistics*, Vol. 39, 1992, pp. 137-152.
26. M. A. Trick, "Scheduling multiple variable-speed machines," *Operations Research*, Vol. 42, 1994, pp. 234-248.
27. M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *USENIX Symposium on Operating Systems Design and Implementation*, 1994, pp. 13-23.
28. The ARM Cooperation, http://www.arm.com.
29. The Transmeta Cooperation, http://www.transmeta.com.
30. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Annual Foundations of Computer Science*, 1995, pp. 374-382.
31. Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," *39th Design Automation Conference*, 2002.
32. D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," *22nd IEEE Real-Time Systems Symposium*, 2001.

Yang Yu is a Ph.D. degree candidate in the department of electrical engineering at the University of Southern California (USC). He received both BS and MS degrees in computer science from Shanghai JiaoTong University in China. His research interests include system synthesis and resource management in real-time environments, networked embedded systems, and sensor networks.

Viktor K. Prasanna is a professor of electrical engineering and computer science at the University of Southern California (USC). He is also a member of the US National Science Foundation supported Integrated Media Systems Center (IMSC) and an associate member of the Center for Applied Mathematical Sciences (CAMS) at USC. His research interests include high performance computing, parallel and distributed systems, network computing, and embedded systems. He received the BS degree in electronics engineering from Bangalore University, the MS degree from the School of Automation, Indian Institute of Science, and the PhD degree in computer science from the Pennsylvania State University. He has published extensively and consulted for industries in the above areas. He is the steering committee cochair of the International Parallel and Distributed Processing Symposium (IPDPS) (merged IEEE International Parallel Processing Symposium (IPPS) and Symposium on Parallel and Distributed Processing (SPDP)). He is the steering committee chair of the International Conference on High Performance Computing (HiPC). He serves on the editorial boards of the *Proceedings of the IEEE* and the *Journal of Parallel and Distributed Computing.* He was the founding chair of the IEEE Computer Society's Technical Committee on Parallel Processing. He is a fellow of the IEEE. He serves as the Editor-in-Chief of the IEEE Transactions on Computers.