

# Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems\*

Yang Yu and Viktor K. Prasanna  
Department of EE-Systems  
University of Southern California  
Los Angeles, CA 90089-2562  
{yangyu, prasanna}@halcyon.usc.edu  
<http://pacman.usc.edu>

## Abstract

*In recent years, power management and power reduction has become a critical issue in portable systems that are designed for real-time use. In this paper, we study the problem of static allocation of a set of independent tasks onto a real-time system consisting of heterogeneous processing elements, each enabled with discrete Dynamic Voltage Scaling. The allocation problem is first formulated as an extended Generalized Assignment Problem. A linearization heuristic (LR-heuristic) is then extended for solving 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 real-time constraints are tight, the LR-heuristic achieves 15% off the optimal energy consumption 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.*

## 1. Introduction

In recent years, power management and power reduction has become increasingly important in various computation and communication systems. It is a critical issue 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 not only on the accuracy of computations, but on their timeliness. The performance requirements as well as

the power-consumption constraints require implementing different parts of the systems in dedicated hardware blocks. As a result, modern real-time systems [13] 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 reside on a single board, yielding heterogeneous multiprocessors that exploit task-level parallelism in applications. Examples of such systems are mobile computing environment [10] and distributed embedded systems [6], among others. Resource allocation and scheduling in such systems is already challenging due to the need to address real-time constraints and system heterogeneity. The problem becomes more challenging when power management and reduction is also considered as one of the design objectives. The power consumption of the system must be carefully balanced against desired system performance.

Power management and reduction can be achieved by two methods: (1) activity based dynamic power management [17], and (2) dynamic supply voltage scaling [21]. 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 consumption. The rationale behind DVS technique is to stretch out task execution time through CPU frequency and voltage reduction. The traditional real-time scheduling theory [13] deals with fixed CPU speed, and hence cannot be directly

---

\*This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-C-00-1633.

applied to this situation. Power-aware scheduling via DVS can be classified as *static* and *dynamic* techniques. Static techniques are applied at design time by allocating and scheduling resources using off-line approaches, while dynamic techniques control the runtime behavior of the systems to reduce power consumption.

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 [24] that can adjust clock frequency from 200 to 700 MHz and corresponding supply voltage from 1.1V to 1.6V, in 33 MHz steps, and (2) the ARM7D processor [22] that can run at 33 MHz (5V supply voltage) and 20 MHz (3.3V supply voltage).

In this paper, we study the problem of static allocation of a set of independent tasks onto a real-time system consisting of heterogeneous processing elements, each equipped with discrete DVS feature. The tasks considered in this paper are assumed to be periodic. In general, this allocation problem is NP-complete. Actually, the problem of scheduling on homogeneous multiprocessors with task preemption allowed to minimize the number of tasks that miss their deadlines is NP-complete [11]. 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 [19]. An extended LR-heuristic [19] is then used for solving 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 consumption 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.

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 is given in Section 7.

## 2. Related Work

For brevity, the term scheduling is used to mean both allocation and scheduling in this section. Based on the underlying system model, research in power-

aware resource scheduling using DVS can be classified into two categories: (1) scheduling on uni-processors, and (2) scheduling in multi-processor systems. There is extensive work targeting uni-processors. However, very little is known about power-aware scheduling in multi-processor systems.

- **Power-aware scheduling on uni-processors:** One of the earliest work, [25], provides an optimal static scheduling algorithm to minimize the total energy consumption while satisfying the relative deadline of all tasks. A non-preemptive variable voltage scheduling heuristic with the assumption of zero delay in changing voltage levels is developed in [7]. An efficient solution for scheduling periodic real-time tasks with (potentially) different power consumption characteristics is presented in [2]. The equivalence of the static scheduling problem to the reward-based scheduling problem with concave reward functions is shown in [3]. Also, an integrated heuristic for dynamic adjustment of discrete voltage values is developed in [15].
- **Power-aware scheduling in multi-processor systems:** For frame-based tasks and homogeneous multi-processor environment, a dynamic processor supply voltage and speed adjustment mechanism using slack reclamation is discussed in [28]. A power-conscious joint scheduling for periodic task graphs and aperiodic tasks in distributed real-time systems is presented in [14]. In this work, the resource allocation is carried out in two steps. A feasible static resource allocation, task assignment and scheduling is first obtained by a system synthesis tool [5], using genetic algorithm. The next step is a re-scheduling of tasks on each individual node. However, no claim of the optimality of the allocation is made. By assuming that task mapping is given, an end-to-end synthesis technique for a set of task pipelines in low power distributed real-time system is presented in [9].

Our research falls into the second category. The contribution of our work is to systematically formulate the power-aware resource allocation problem in real-time system as an extended Generalized Assignment Problem (GAP). By exploring the inherent similarity between these two problems, techniques that are available for solving GAP can be extended to solve power-aware scheduling problems. This paper also serves as a step towards power-aware scheduling of tasks with dependency constraints and communication cost. [14] studies a related problem by assuming continuous DVS techniques. It proposes an empirical two-step approach.

### 3. Problem Definition

**System Model:** The system consists of a set of  $m$  PEs,  $\{PE_1, PE_2, \dots, PE_m\}$ . 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  $PE_k, k = 1, 2, \dots, m$ . Also, the Earliest Deadline First (EDF) [12] scheduling policy is assumed to be used by each PE.

**Application Model:** A set of  $n$  independent periodic real-time tasks,  $\{T_1, T_2, \dots, T_n\}$ , 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  $T_i$  [12]. This means each instance of a task must complete execution before starting the next instance of the same task. The workload of a task is measured by the worst-case number of CPU cycles required for executing the task, which can be different on different PEs, due to the 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  $S_{ik}$  (given in cycles per second) is  $t_{ik} = \frac{C_{ik}}{S_{ik}}$ . Because the number of voltage levels of  $PE_k$  is  $V_k$ ,  $S_{ik}$  can take at most  $V_k$  discrete values.

In each PE, the voltage is assumed to be dynamically switched, 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  $\sum_{j=1}^n \lceil \frac{P_i}{P_j} \rceil$ . 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  $S_{ik}$  is denoted by  $g_{ik}(S_{ik})$ , a strictly increasing convex function, represented by a polynomial of at least second degree [8]. If  $T_i$  occupies the CPU of  $PE_k$  during the time interval  $[t_1, t_2]$ , the energy consumed by  $PE_k$  during this interval is given by  $E_{ik}(t_1, t_2) = \int_{t_1}^{t_2} g_{ik}(S_{ik}(t)) dt$ . If the CPU speed,  $S_{ik}$ , during the execution of  $T_i$  is fixed,  $E_{ik}$  can be calculated as  $g_{ik}(S_{ik})t_{ik}$ . 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.

**Task Scheduling and Resource Allocation:** A schedule of a set of periodic tasks on  $PE_k$  is feasible if each instance of task  $T_i$  in the set can be assigned at least  $C_{ik}$  CPU cycles before its relative deadline. The utilization of a task on a PE is defined as the CPU time that the task needs for completing its execution, normalized by the period of the task. Let  $u_{ik}$  denote the utilization of  $T_i$  on  $PE_k$ . If  $T_i$  is executed at speed  $S_{ik}$ , we have  $u_{ik} = \frac{t_{ik}}{P_i} = \frac{C_{ik}}{S_{ik}P_i}$ . According to [12], the

EDF schedule of a set of tasks,  $T$ , to be executed on  $PE_k$  is *feasible* if and only if the total utilization of all tasks in  $T$  does not exceed the computation capacity of  $PE_k$ , i.e.  $\sum_{T_i \in T} u_{ik} \leq 1$ .

A resource allocation is defined as a mapping of all the tasks onto the PEs, along with the setting of voltage level for each task on the corresponding PE. Each task can be mapped onto exactly one PE and can be executed with a fixed voltage level on that PE. An allocation is feasible if for every  $PE_k$ , the EDF schedule of tasks allocated on  $PE_k$  is feasible. A feasible allocation is optimal if the overall energy consumption of the system is minimal among all feasible allocations. Because different tasks may have different periods, the overall energy consumption is calculated as the energy consumption of the system during the least common multiple of periods of all tasks, denoted by  $LCM$ .

### 4. Integer Linear Programming Formulation

The problem defined in Section 3 essentially asks for a static allocation of tasks onto voltage levels. 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  $j$ -th 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  $T_i$  is executed on voltage level  $VL_j$ . Similarly, let  $e_{ij}$  denote the energy consumption in executing an instance of  $T_i$  on  $VL_j$ . Since we are optimizing the system energy consumption during each  $LCM$ , let  $e'_{ij}$  denote the total energy consumption for executing  $T_i$  on  $VL_j$  during a  $LCM$ . We have  $e'_{ij} = e_{ij} \frac{LCM}{P_i}$ . Given an allocation, the utilization of voltage level  $VL_j$ ,  $U_j$ , is defined as the sum of the utilization of tasks that are mapped onto  $VL_j$ . For any feasible allocation, we have  $\sum_{VL_j \in VLG(k)} U_j \leq 1, k = 1, \dots, m$ . Let  $\{x_{ij}\}$  be a set of 0-1 variables such that  $x_{ij}$  equals one if  $T_i$  is mapped on  $VL_j$ , and zero otherwise. The problem can now be formulated as the following ILP problem, ILP(1).

$$\begin{aligned}
 & \text{Minimize} && \sum_{i=1}^n \sum_{j=1}^{\hat{m}} e'_{ij} x_{ij} \\
 & \text{subject to} && \\
 & && \sum_{i=1}^n u'_{ij} x_{ij} - U_j \leq 0 && j = 1, 2, \dots, \hat{m} && (1) \\
 & && \sum_{VL_j \in VLG(k)} U_j \leq 1 && k = 1, 2, \dots, m && (2) \\
 & && \sum_{j=1}^{\hat{m}} x_{ij} = 1 && i = 1, 2, \dots, n && (3) \\
 & && x_{ij} \in \{0, 1\} && i = 1, 2, \dots, n, && j = 1, 2, \dots, \hat{m} && (4)
 \end{aligned}$$

This formulation is in the same form of a Generalized Assignment Problem (GAP) [19] except for constraints (2) above. The only difference between ILP(1) and GAP is that the capacity of resources, in terms of the utilization of voltage levels, are defined in groups in ILP(1), whereas in case of GAP, they are defined individually. If there is a feasible solution for ILP(1), a GAP formulation can be obtained by substituting the  $U_j$ 's with their corresponding values in the solution and removing constraints (2). Intuitively, the value 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 show an upper bound on the number of split tasks (to be defined later) in a basic solution [16] of LP(1). A linear relaxation heuristic proposed in [19] for solving GAP is then extended to solve ILP(1). Finally, an upper bound of 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 mapped*, 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 any solution of LP(1). Due to the similarity between ILP(1) and GAP, it can be easily shown that the number of split jobs in a basic solution [16] to the linear relaxation LP(1) is at most the number of voltage levels allocated to capacity (Theorem 1 in [19]), which is  $\hat{m}$  in the worst case. However, because of the special properties introduced by constraints (2) in LP(1), we show an alternative formulation of the problem, and consequently improve the bound to  $m$ .

**Lemma 5.1** *In every basic solution of LP(1), the number of split tasks is at most 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 form the following alternative linear programming formulation, LP(2):

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

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 value of  $U_j$ 's can be determined and thus form a basic solution to LP(1).

Given a basic solution to LP(2), the utilization of task  $T_i$  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  $y_{ik} + y'_{ik} = \sum_{VL_j \in VLG(k)} x_{ij}$ . We then consider the following linear programming formulation, LP(3), for which a feasible solution is desired:

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

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 [20]. Any basic solution to LP(3) is known to have the property that the number of split tasks is at most the number of PEs allocated to capacity [20], and thus the claim follows.  $\square$

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 mapping and assuming that  $VL_j$  belongs to  $VLG(k)$ , the value of  $U_j^{max}$  can be calculated by subtracting from 1 the sum of the utilization of tasks that are integrally mapped onto  $PE_k$ . Specifically, we have  $U_j^{max} = 1 - \sum_{VL_l \in VLG(k)} \sum_{x_{il}=1} e_{il}$ . A variable  $x_{ij}$  is defined to be *useless* if  $u'_{ij} > U_j^{max}$ . Here, the definition of useless variable is an extension of its original definition in [19], due to constraints(2) in ILP(1). However, the linear relaxation heuristic, LR-heuristic, proposed in [19] can still be used to solve ILP(1). The LR-heuristic is as follows:

**Input:** an instance of ILP(1)

**Output:** a mapping of tasks onto 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, deleting the corresponding tasks and updating the capacity of PEs.

Step 3) Go to Step 0.

Initially, the values of  $U_j^{max}$ 's are set to 1, for  $j = 1, 2, \dots, \hat{m}$ . Once a (partial) mapping is obtained by executing Step 1) at least once, the values of  $U_j^{max}$ 's are

---

**Begin**

1.  $T^* = T$
2. Set  $U_j^{max} = 1$ , for  $j = 1, 2, \dots, \hat{m}$
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 voltage levels and  $u'_{ij} \leq U_j^{max}$ . Record the information as tuple  $(t_i, VL_j, e'_{ij})$
5. Select the tuple that gives the minimal value of  $e'_{ij}$
6. Allocate the task in the selected tuple to the voltage level in that tuple and remove the task from  $T^*$
7. Update  $U_j^{max}$  of all voltage levels

**End**

---

**Figure 1. Pseudo code for Greedy heuristic**

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 [19]) 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 [20].

## 6. Experimental Results

In this section, we demonstrate the quality of solutions produced by LR-heuristic in our experiments. A simulator based on the system and application models presented in Section 3 was developed to evaluate the performance of the LR-heuristic for solving ILP(1) problems 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 tightness of the constraints, the number of voltage levels per PE) on the performance of the LR-heuristic. Due to space limitation, we illustrate the results for a small set of parameter values. Additional results can be found in [27].

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 [4]. The original objective function of min-min heuristic is to minimize the *makespan* [4] of a set of tasks. Here, the heuristic is modified so that the overall energy consumption of the system is minimized, while satisfying the utilization constraint of each PE. The pseudo code for the Greedy is shown in Figure 1.

For small size problems, the number of PEs was fixed at 5 (so that the optimal solution can be computed in reasonable amount of time by using LINDO), while the number of tasks varied from 20 to 40. The system heterogeneity is captured by the distribution of the worst-case number of CPU cycles ( $C_{ik}$ ) required by different tasks on different PEs. It can be characterized by task and PE heterogeneities. To emphasize the impact on system performance due to high heterogeneity, the results for high task and PE heterogeneities is illustrated in this paper. Let  $\mathcal{C}$  denote the matrix composed by  $\{C_{ik}\}$ , where  $i = 1, 2, \dots, n$  and  $k = 1, 2, \dots, m$ . The  $\mathcal{C}$  matrix was generated using a Gamma distribution based method [1]. The mean value along the task axis,  $\mu_{task}$  was set to 200. Other two 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  $X = \frac{n}{m}$ , where  $n$  is the number of tasks and  $m$  is the number of PEs.  $P_i$  is calculated as  $f \cdot w_i \cdot X$ , where  $f$  is a pre-specified positive value for adjusting the tightness of the real-time constraints (i.e., the relative deadlines of tasks). To set the value of  $LCM$  within some reasonable range, the value of  $P_i$  was adjusted to some close value such that the value of  $LCM$  is at most 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 is assumed that other CPU speeds are distributed uniformly between the maximum and minimum speeds. Therefore, the other two levels of CPU speed are set to 0.75 and 0.5. The energy function of  $T_i$  on  $PE_k$ ,  $g_{ik}(S)$ , was of the form  $a_{ik} \cdot S^{b_{ik}}$ , where  $S$  is the CPU speed of  $PE_k$  and  $a_{ik}$  and  $b_{ik}$  were random variables with uniform distribution between 2 and 10, and 2 and 3, respectively [15].

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.

### 6.1 Results

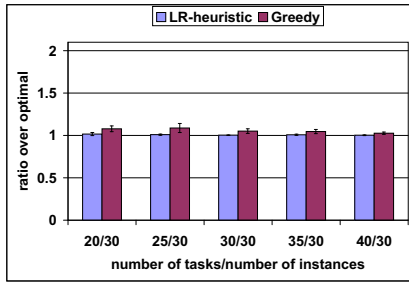
The experimental results for small size problems when  $f$  is set to 2.5, 2.0, and 1.5 are shown in Fig-

**Table 1. The miss rate and the average number of unallocated tasks of LR-heuristic and Greedy**

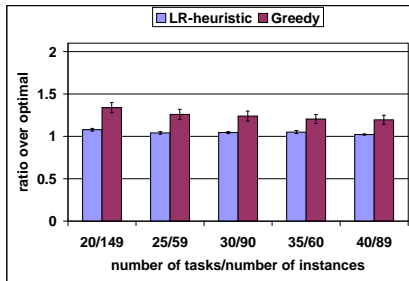
$f$		2.0					1.5				
# of tasks		20	25	30	35	40	20	25	30	35	40
miss rate (%)	LR-heuristic	0	0	0	0	0	0	0	0	0	0
	Greedy	1	2	0	0	1	40	49	59	72	73
unallocated tasks	LR-heuristic	0	0	0	0	0	0	0	0	0	0
	Greedy	0.01	0.03	0	0	0.01	0.69	1.04	1.48	2.21	2.34

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

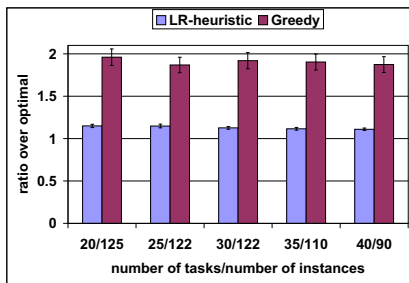
problem size	5 PEs, $f = 2.0$					10 PEs, $f = 1.6$				
# of tasks	20	25	30	35	40	60	70	80	90	100
LINDO	0.28	0.26	0.36	0.51	0.37	-	-	-	-	-
LR-heuristic	0.071	0.071	0.078	0.077	0.078	0.26	0.30	0.34	0.37	0.40
Greedy	0.0004	0.0007	0.0009	0.0013	0.0016	0.007	0.009	0.012	0.015	0.02



(a)  $f = 2.5$



(b)  $f = 2.0$

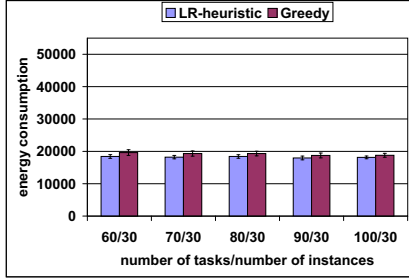
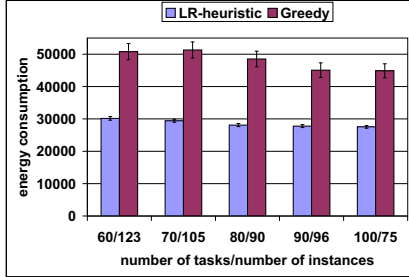


(c)  $f = 1.5$

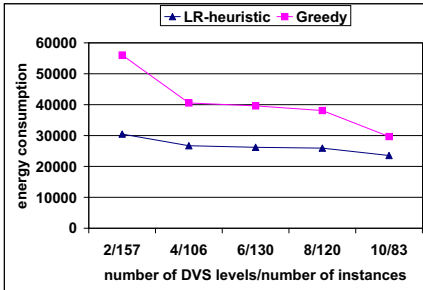
**Figure 2. Experimental results for small size problems**

ure 2. It shows the ratio of the overall system energy consumption 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 succeed 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 has a 95% confidence interval with a 5% (or better) precision [23]. The number of instances may 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 calculation 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 the LR-heuristic always outperforms the Greedy. When  $f = 2.5$ , both heuristics perform quite well. When the value of  $f$  decreases (i.e., the real-time constraints become tighter), the performance of both heuristics become worse. However, the performance of LR-heuristic is still quite acceptable when  $f = 1.5$ , achieving 15% off the optimal, while the performance of Greedy is around 90% off the optimal.

It was observed that when  $f$  decreases, the number of instances for which the Greedy fails to find a feasible allocation increases rapidly. On the contrary, the LR-heuristic still succeeded in all instances even when  $f = 1.5$ . Table 1 shows the miss rate (the number of instances that a heuristic fails, normalized with respect to the total number of instances) and the average number of unallocated tasks over all instances of both heuristics when  $f = 2.0$  and 1.5. The astonishingly high miss rate of the Greedy when  $f = 1.5$  indicates the inappropriateness of Greedy for tightly constrained problems.

(a)  $f = 1.6$ (b)  $f = 1.1$ 

**Figure 3. Experimental results for large size problems**



**Figure 4. Performance of heuristics as a function of the number of DVS levels (10 PEs, 80 tasks,  $f = 1.2$ )**

Figure 3 shows the relative performance of LR-heuristic and Greedy for large size problems when  $f = 1.6$  and  $1.1$ . Again, the number of instances for each case was chosen to be large enough such that the presented data has a 95% confidence interval with a 5% (or better) precision. Similar trends in performance was observed when the value of  $f$  decreases. When  $f$  equals  $1.1$ , the LR-heuristic showed up-to 40% improvement over the performance of Greedy. It was noticed that for the same value of  $f$ , the relative performance of Greedy is much better for large size problems, compared with small size problems. This may be due to the fact that for the same value of  $f$ , 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 4 DVS levels in small size problems, and (2) for large size problems, the size of  $\mathcal{C}$  matrices increases and hence leads to a higher likelihood to get 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.

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 Figure 4. The performance of both heuristics improves when the number of DVS levels increases. Additionally, LR-heuristic is 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. It shows that when the number of tasks increases, the ratio of running time of LR-heuristic to that of Greedy decreases 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 a off-line design phase problem where running time is not a critical concern, LR-heuristic is a good choice for solving the problem.

## 7. Concluding Remarks

This paper discussed power-aware resource allocation for independent tasks in heterogeneous real-time systems. The problem was formulated as an extended Generalized Assignment Problem and was solved by an extension of the LR-heuristic. In the future, we plan to study some relative 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 the communication is carried out in an ad hoc fashion. A ILP formulation based solution for this problem is proposed in [26]. Convex optimization techniques may be used to solve the second class of problems. Also, we are interested in designing on-line power-aware scheduling policies for distributed real-time systems based on the techniques developed in this paper.

This work is performed as 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] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. task execution time modeling for heterogeneous computing systems. In *9th Heterogeneous Computing Workshop*, pages 185–199, May 2000.
- [2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *13th Euromicro Conference on Real-Time Systems (ECRTS01)*, June 2001.
- [3] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *22nd Real-Time Systems Symposium*, Dec. 2001.
- [4] T. D. Braun, S. Ali, H. J. Siegel, and A. A. Maciejewski. Using the Min-Min heuristic to map tasks onto heterogeneous high-performance computing systems. In *2nd Symposium of the Los Alamos Computer Science Institute*, Oct. 2001.
- [5] R. P. Dick and N. K. Jha. MOCSYN: Multiobjective core-based single-chip system synthesis. In *Design Automation & Test in Europe Conference*, pages 263–270, Mar. 1999.
- [6] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, NJ, 1994.
- [7] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *35th Design Automation Conference*, pages 176–181, June 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 *19th IEEE Real-Time Systems Symposium (RTSS'98)*, Dec. 1998.
- [9] D.-I. Kang, S. P. Crago, and J. Suh. Power-aware design synthesis techniques for distributed real-time systems. In *ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001)*, June 2001.
- [10] M. J. Kumar and S. Venkatesh. Power saving schemes for mobile computing environment. In *4th International Conference on Advanced Computing*, pages 296–302, Dec. 1996.
- [11] E. L. Lawler. *Mathematical Programming: The State of the Art*, chapter Recent results in the theory of machine scheduling, pages 202–233. Springer-Verlag, 1983.
- [12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of ACM*, 20(1):46–61, 1973.
- [13] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.
- [14] J. Luo and N. K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Computer-Aided Design*, pages 357–364, Nov. 2000.
- [15] P. Mejía-Alvarez, E. Levner, and D. Mossé. An integrated heuristic approach to power-aware real-time scheduling. In *Workshop on Power-Aware Computer Systems (PACS'02)*, Feb. 2002.
- [16] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice-Hall, NJ, 1982.
- [17] Q. Qiu and M. Pedram. Dynamic power management based on continuous-time Markov decision processes. In *36th Design Automation Conference*, pages 555–561, June 1999.
- [18] L. Schrage. *Linear, Integer, and Quadratic Programming with LINDO*. The Scientific Press, Redwood city, CA, 1986.
- [19] M. A. Trick. A linear relaxation heuristic for the generalized assignment problem. *Naval Research Logistics*, 39:137–152, 1992.
- [20] M. A. Trick. Scheduling multiple variable-speed machines. *Operations Research*, 42:234–248, 1994.
- [21] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, Nov. 1994.
- [22] <http://www.arm.com>.
- [23] <http://www.mathworld.com>.
- [24] <http://www.transmeta.com>.
- [25] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.
- [26] Y. Yu and V. K. Prasanna. An integer linear programming based approach for power-aware task allocation in real-time distributed embedded systems. in preparation.
- [27] Y. Yu and V. K. Prasanna. Power-aware resource allocation in distributed real-time systems. Technical report, University of Southern California, 2002. in preparation.
- [28] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *22nd IEEE Real-Time Systems Symposium*, Dec. 2001.