Energy-Aware Partitioning for Multiprocessor Real-Time Systems

Hakan Aydin, Qi Yang
Computer Science Department
George Mason University
Fairfax, VA 22030
(aydin, qyang1)@cs.gmu.edu

Abstract

In this paper, we address the problem of partitioning periodic real-time tasks in a multiprocessor platform by considering both feasibility and energy-awareness perspectives: our objective is to compute the feasible partitioning that results in minimum energy consumption on multiple identical processors by using variable voltage Earliest-Deadline-First scheduling. We show that the problem is NP-Hard in the strong sense on $m \geq 2$ processors even when feasibility is guaranteed a priori. *Then, we develop our framework where* load balancing *plays a major role in producing energy-efficient partitionings. We evaluate the feasibility and energy-efficiency performances of partitioning heuristics experimentally.*

1 Introduction

Multiprocessor scheduling of periodic tasks is one of the most extensively studied areas in real-time systems research. In general, the approaches fall into either *global* or *partitioning-based* scheduling categories. In global scheduling [1, 7, 11], there is a single *ready queue* and task migrations among processors are allowed. In contrast, partitioning-based approach [4, 9, 13, 16] allocates each task to one processor permanently (thus, task migrations are not allowed) and resorts to well-established single-processor scheduling policies to guarantee the feasibility.

In recent years, we witnessed the emergence of *low-power computing* as a prominent research area in many Computer Science/Engineering disciplines. The main low-power computing technique in real-time systems has been *variable voltage scheduling (or, dynamic voltage scaling)* [2, 3, 10, 17, 19]. The technique hinges upon the speed/voltage adjustment capability of state-of-the-art microprocessors and exploits the convex relationship between CPU speed and power consumption. In principle, it is possible to obtain striking (usually, quadratic) energy savings by reducing CPU speed. On the other hand, the feasibility of the schedule must be preserved even with reduced speed and this gives rise to the problem of *minimizing energy consumption while meeting the deadlines*.

In [3], three complementary dimensions of real-time variable voltage scheduling were identified: At *static* level, task-level optimal speed assignments are computed assuming a worst-case workload. Since tasks usually complete earlier than what is predicted in the worst-case scenario, on-line adjustments on pre-computed static speeds can provide additional savings. Thus in addition, we have *dynamic* and *speculative* dimensions: These dimensions address how to *reclaim* and *predict/provision for* unused computation time, respectively. As one recent study shows [15], the near-optimal performance of various techniques proposed for *single processor* variable voltage real-time scheduling demonstrate a level of maturity for the area.

A few multiprocessor power-aware scheduling techniques have been recently proposed by research community. However, these usually consider *aperiodic* task sets: Gruian [9] addressed non-preemptive scheduling of tasks on multiprocessor systems. Zhu et al. proposed a run-time slack reclamation scheme for tasks sharing a single, global deadline [20]. This was followed by another paper [21] where the same authors extend the model to aperiodic tasks with precedence constraints. Yang et al. proposed a two-phase scheduling scheme for system-on-chip with two processors [18]. To the best of our knowledge, the only research effort that combines *periodic* multiprocessor real-time scheduling with energy-awareness is a study by Funk, Goossens and Baruah: in [6], the authors address the problem of determining optimal speed/voltage level selection for *global* Earliest Deadline First (EDF) scheduling.

In this paper, we address the problem of energy-minimization through dynamic voltage scaling in the context of *partitioning-based* approaches. Global and partitioning-based approaches are known to have their own advantages and disadvantages in traditional (i.e. non-power-aware, constant-speed) multiprocessor real-time scheduling [13, 7]. From energy-awareness perspective, the immediate advantage of concentrating on partitioning-based approaches is the ability to apply well-established uniprocessor variable voltage scheduling techniques in all three (i.e. static, dynamic and speculative) levels once task-to-processor assignments are determined. In particular, our task assignment strategies will make their decisions using worst-case workload information and therefore will determine the *optimal* static speed assignments on each processor. After this point, dynamic and speculation reclamation strategies [3] can be applied on each processor to further exploit energy-saving opportunities at run-time.

Partitioning-based multiprocessor real-time scheduling considers *feasibility* as the main objective. The problem is invariably NP-Hard [8, 13] and appears in two variations: Minimizing the number of processors needed to guarantee the feasibility of the task set, or alternatively, given a *fixed* multiprocessor platform, finding sufficient schedulability (usually, utilization) bounds. Our work opts for the second setting, thus we assume the existence of a given number of processors. Considering the intractable nature of the problem, several heuristics and their performance analysis were subject of many research papers, including First-Fit, Best-Fit, Next-Fit and Worst-Fit [13, 4, 16]. In fact, when using Earliest Deadline First scheduling, the problem has a close affinity with Bin-Packing [8, 5], and the results/heuristics available in this widely-studied area provide insights for partitioning-based scheduling.

When we add the *energy* dimension to the problem, we may need to modify/expand performance metric accordingly. In fact, as we show later in the paper, some heuristics have very good (albeit not optimal) feasibility performances even at high utilizations, but result in poor energy performance. Yet some others have excellent energy performance at low utilizations, but their feasibility performances degrade rapidly with increasing load. Thus, we propose a metric to capture both dimensions of power-aware multiprocessor real-time scheduling: *Timeliness/Energy* metric favors in general the heuristics with high feasibility and low energy consumption performances, giving a more accurate measure of overall performance.

After giving the system model in Section 2, we formalize the problem and justify our decision to commit to EDF at each processor from both feasibility and energy points of view, for any task-to-processor assignment in Section 3. We establish that the problem is NP-Hard in the strong sense. Given many intractability results regarding multiprocessor real-time scheduling, this is only to be expected; but we show that the problem of minimizing energy-consumption on partitioned systems remains NP-Hard *even when* the feasibility is guaranteed a priori (by focusing on task sets that can be scheduled on a *single* processor in a feasible manner).

Then in Section 4, we characterize the energy-efficient task-to-processor assignment problem as a *load balancing* problem. We introduce the concepts of *balanced* and *unbalanced* task assignments as a way of addressing/assessing the energy-efficiency issues in multiprocessor platforms. Thanks to this characterization, we prove that a partitioning that yields perfectly balanced load necessarily minimizes the total energy consumption. Further, we show that "heavy" tasks with large utilization values must be allocated to separate processors in the optimal solution. In Section 5, we present and comment on the performance of heuristics for the problem. Our analysis distinguishes two cases: In the first one, the scheduler is allowed to order tasks according to (non-increasing) utilization values before running the heuristic, while it is not allowed to do so in the second. It is known that worst-case and average-case performance of algorithms improve when this pre-ordering is allowed [5, 14]. We experimentally show that Worst-Fit-Decreasing (WFD) algorithm dominates other techniques. Finally, for the case where tasks are not ordered according to the utilization values, we show that none of the well-known heuristics offers a clear advantage. We present an efficient heuristic called RESERVATION that combines ideas and results developed in previous sections. The performance of RESERVATION is justified experimentally against other heuristics, before concluding the paper.

2 System Model

We consider the scheduling of a periodic real-time task set $\mathcal{T} = \{T_1, \ldots, T_n\}$ on a set of processors $\mathcal{M} = \{M_1, \ldots, M_m\}$. The period of T_i is denoted by P_i, which is also equal to the relative deadline of the current invocation. All tasks are assumed to be independent and ready simultaneously at $t = 0$.

We assume that all m processors are *identical* from both processing power and speed/energy consumption relationship aspects. Each processor M_i has variable voltage/speed feature, hence its speed S^i (in terms of processor cycles per unit time) can vary between 0 and an upper bound S_{max}. For convenience, we normalize the CPU speed with respect to S_{max}, that is, we assume that $S_{max} = 1.0$.

The power consumption of the processor under the speed S is given by $g(S)$. In current variable-voltage processor technologies, the function $g(S)$ is assumed to be a strictly convex and increasing function on non-negative real numbers [10, 2, 3, 17]. Further, it is usually represented by a polynomial of at least second degree [10, 3]. If the speed of the processor M_i during the time interval $[t_1, t_2]$ is given by $S(t)$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g(S(t))dt$.

In variable voltage/speed settings, the indicator of task-level worst-case workload is the worst-case number of processor cycles required by the task T_i and it is denoted by C_i [3]. Thus, under a constant speed S, the (worst-case) execution time of one instance of T_i is $\frac{C_i}{S}$.

The (worst-case) utilization of task T_i under maximum speed $S_{max} = 1$ is $u_i = \frac{C_i}{P_i}$. We define U_{tot} as the total utilization of the task set $\mathcal{T}$ under maximum speed $S_{max} = 1$, that is, $U_{tot} = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{P_i}$. Note that a necessary condition to have a feasible schedule on m processors is to have $U_{tot} \leq m$, and we will make this assumption throughout the paper.

Finally, given a task-to-processor assignment[1] Π, we will

[1] We use the terms "task(-to-processor) assignment" and "partitioning" interchangeably throughout the paper.

denote the utilization of processor M_i under $S_{max} = 1.0$ by $U_i(\Pi)$ (or simply U_i when the context is clear). Since each task must be assigned to exactly one processor, it is clear that $U_{tot} = \sum_{i=1}^{n} u_i = \sum_{i=1}^{m} U_i$, for any task assignment.

3 Energy Minimization with Partitioning

Our aim in this research effort is to address the following energy-aware real-time scheduling problem (denoted by POWER-PARTITION)

> Given a set $\mathcal{T}$ of periodic real-time tasks and a set $\mathcal{M}$ of m identical processors, find a task-to-processor assignment and compute task-level speeds on each processor such that:
>
> 1. the tasks assigned to each processor can be scheduled in a feasible manner, and
> 2. the total energy consumption of $\mathcal{M}$ is minimum (among all feasible task allocations)

At this point, it can be observed that POWER-PARTITION is NP-Hard in the strong sense: Suppose that there exists a polynomial time algorithm that produces a *feasible* assignment of real-time tasks with minimum energy consumption, and NIL if no feasible partitioning exists. Since checking the feasibility of a set of real-time tasks on a multiprocessor platform even with a single, overall deadline (and by using the maximum speed S_{max}) is NP-Hard in the strong sense and the supposedly polynomial-time algorithm would solve this problem as well, POWER-PARTITION is NP-Hard in the strong sense.

Given any task assignment Π, consider the scheduling policy and speed assignments to be adopted on each processor. The classical result by Liu and Layland [12] implies that Earliest Deadline First (EDF) scheduling policy is optimal from the feasibility point of view. In addition, the following result (adapted from [3]) establishes that EDF is also optimal from energy consumption point of view when used with a constant speed equal to the utilization of the task set assigned to that processor.

Proposition 1 (from [3]) *Consider a single processor system and a set of periodic real-time tasks with total utilization $U_{tot} \leq 1.0$. The optimal speed to minimize the total energy consumption while meeting all the deadlines is constant and equal to $\bar{S} = U_{tot}$. Moreover, when used along with this speed $\bar{S}$, any periodic hard real-time scheduling policy which can fully utilize the processor (e.g., Earliest Deadline First) can be used to obtain a feasible schedule.*

In short, for a given task assignment, we can safely commit to EDF with constant speed $\bar{S} = U_i$ on processor M_i without compromising feasibility or energy-efficiency, where $U_i \leq 1.0$ is the total utilization (load) of tasks assigned to M_i. Note that if U_i exceeds 1.0 for a given processor, it is impossible to meet the deadlines even with the maximum CPU speed, hence the task assignment under consideration is infeasible.

Also, note that we need to specify the interval of time during which we aim to minimize energy consumption. In accordance with [3] and considering that the schedules on *all* processors can be repeated at every *hyperperiod* P without hurting feasibility or energy-efficiency, we focus on minimizing energy consumption during $P = lcm(P_1, \ldots, P_n)$. The energy consumption of task T_j running on processor M in interval $[0, P]$ when executed with constant speed S is given by: $g(S) \cdot \frac{P}{P_j} \cdot \frac{C_j}{S}$. The energy consumption of all tasks allocated to the processor M_i is therefore:

$$E(M_i) = \sum_{T_j \ assigned \ to \ M_i} g(S) \cdot \frac{P}{P_j} \cdot \frac{C_j}{S} \tag{1}$$

When we substitute the optimal speed expression $\bar{S} = \sum_{T_j \ assigned \ to \ M_i} \frac{C_j}{P_j} = U_i$ (from Proposition 1 and the definition of U_i) for S above, we find the minimum energy consumption on processor M_i as

$$E^*(M_i) = P \cdot g(\bar{S}) \cdot \frac{U_i}{\bar{S}} = P \cdot g(U_i) \tag{2}$$

Considering that P is a constant, independent of assignment and scheduling algorithm, we can now present the optimization problem which is equivalent to POWER-PARTITION:

> Given a set $\mathcal{T}$ of periodic real-time tasks and a set $\mathcal{M}$ of m identical processors, allocate tasks to processors so as to:
>
> $$\text{minimize} \quad \sum_{i=1}^{m} g(U_i) \tag{3}$$
>
> $$\text{subject to} \quad 0 \leq U_i \leq 1.0 \quad i = 1, \ldots, m \tag{4}$$
>
> $$\sum_{j=1}^{m} U_j = U_{tot} \tag{5}$$
>
> where U_i is the total utilization (load) of the processor P_i after task allocation and $g()$ is the power consumption function (convex and strictly increasing).

Definition 1 *The task assignment (partitioning) that yields the minimum overall energy consumption is called the* power-optimal *assignment (partitioning).*

Motivational Example: Consider three tasks with $u_1 = 0.5, u_2 = 0.25$ and $u_3 = 0.15$ to be executed on $m = 2$ identical processors[2]. Assume the power consumption function $g(S) = S^2$. It is not difficult to see that *any* assignment of these tasks to two processors yields a feasible schedule under EDF. If we ignore symetrical allocations, we have only four possible partitionings:

[2]For simplicity, assume that all the periods are equal to 1.

1. All three tasks are allocated to one processor (Figure 1, left): Energy consumption= $0.9^2 = 0.81$.

2. T_1 and T_2 are allocated to one processor and T_3 is allocated to the other processor (Figure 1, right): Energy consumption = $0.75^2 + 0.15^2 = 0.585$.

3. T_1 and T_3 are allocated to one processor and T_2 is allocated to the other processor (Figure 2, left): Energy consumption = $0.65^2 + 0.25^2 = 0.485$.

4. T_2 and T_3 are allocated to one processor and T_1 is allocated to the other processor (Figure 2, right): Energy consumption = $0.4^2 + 0.5^2 = 0.41$.

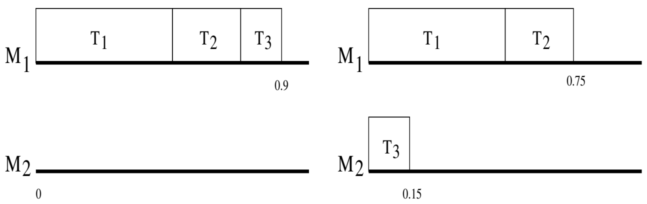

Figure 1. Task Assignment Options 1 (left) and 2 (right)

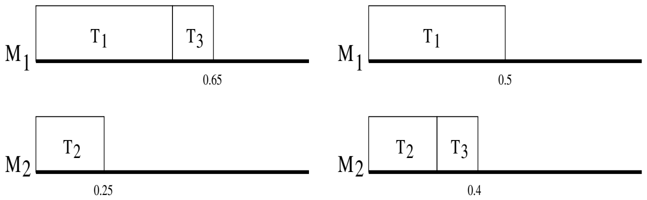

Figure 2. Task Assignment Options 3 (left) and 4 (right)

This simple example with two processors illustrates that energy characteristics of *feasible* partitions can differ greatly: the most energy efficient task assignment consumes just half of the energy consumed by the first partition. In addition, we observe that the best choice in this example turns out to be the one which yields the most "balanced" partitioning (load) on two processors. In fact, it is possible to show the following:

Proposition 2 *A task assignment that* evenly *divides the total load U_{tot} among all the processors, if it exists, will minimize the total energy consumption for any number of tasks.*

Proof: Follows from the strictly convex nature of power consumption function $g()$: the function $\sum_{i=1}^{m} g(U_i)$ is also strictly convex and minimizing it subject to $0 \leq U_i \leq 1.0$ and $\sum_{j=1}^{m} U_j = U_{tot}$ would yield $U_i = \frac{U_{tot}}{m}$. Further, this is the unique global minimum. Hence, *if* there exists a task assignment resulting in a perfectly balanced load, this achieves minimum overall energy consumption.

□

As discussed previously, looking for a feasible task-to-processor assignment is NP-Hard in the strong sense, which implies the same for POWER-PARTITION given by (3), (4) and (5). Interestingly, with the help of Proposotion 2 it is possible to prove a stronger result: *the problem remains NP-Hard in the strong sense even if the task set is guaranteed to be feasible with a total utilization not exceeding 1.0.* In this case, any *reasonable* [13, 14] task allocation algorithm (such as First-Fit, Best-Fit, Worst-Fit or even Random-Fit) would produce a feasible partitioning in linear time, but computing the partitioning that minimizes overall energy consumption is intractable.

Theorem 1 *POWER-PARTITION is NP-Hard in the strong sense on $m \geq 2$ processors for trivially-schedulable task sets with $U_{tot} \leq 1.0$.*

Proof: We will reduce 3-PARTITION problem which is known to be NP-Hard in the strong sense [8] to POWER-PARTITION problem.
3-PARTITION: Given a set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ integers, a bound B, a size $s(a_i) \in Z^+$ for each a_i where $B/4 < s(a_i) < B/2$ and $\sum_{i=1}^{3m} s(a_i) = mB$, can A be partitioned into m disjoint subsets $A_1, \ldots, A_m$ such that the sum of elements in each subset is exactly B?

Suppose that there exists a polynomial-time algorithm to solve an instance of POWER-PARTITION problem on $m \geq 2$ processors for task sets with $U_{tot} \leq 1.0$. Given an instance of 3-PARTITION problem, we construct the following instance of POWER-PARTITION: we have m processors and the task set $\mathcal{T} = \{T_1, \ldots, T_{3m}\}$ where $C_i = s(a_i)$ and $P_i = mB$ for each task T_i. Observe that $U_{tot} = \sum_{i=1}^{3m} \frac{C_i}{P_i} = \frac{mB}{mB} \leq 1.0$. The power consumption function $g()$ is strictly convex and increasing on non-negative real numbers.

Now, invoke POWER-PARTITION problem and compute (by assumption, in polynomial-time) the energy consumption E^* of power-optimal partitioning. We claim that the answer to the corresponding instance of 3-PARTITION problem is "yes" if and only if $E^* = mg(\frac{1}{m})$.

The 3-PARTITION instance admits a "Yes" answer if and only if the summation of elements in each subset A_i is exactly B, in other words if and only there if exists a "perfectly balanced" partitioning of elements in A into m disjoint subsets. Proposition 2 implies that $E^* \geq mg(\frac{U_{tot}}{m})$, and further $E^* = mg(\frac{U_{tot}}{m})$ if and only if there exists a perfectly balanced partitioning of tasks to m processors with $U_i = \frac{U_{tot}}{m}$ $i = 1, \ldots, m$. Since $U_{tot} = 1.0$ in our problem, if $E^* = mg(\frac{1}{m})$, then there exists a perfectly balanced partitioning. But if this is the case, in the corresponding 3-PARTITION problem, the sum of elements at each subset A_i (matching the processor M_i in POWER-PARTITION) is exactly B and the instance admits a "Yes" answer. Conversely, if $E^* > mg(\frac{U_{tot}}{m})$ in POWER-PARTITION instance, then there exists no perfectly balanced partitioning with $U_i = \frac{1}{m} = \frac{B}{mB}$, and 3-PARTITION instance has a "No" answer. □

4 Load Balancing For Energy Efficiency

Given the inherent intractability of the problem, we must look for heuristics. However, before giving the performance evaluation of heuristics, we present *balanced* and *unbalanced* partitioning (or, task assignment) concepts as instruments to understand and address energy-efficiency issues in multiprocessor platforms.

Definition 2 *A task-to-processor assignment* Π *is said to be* unbalanced *if the total energy consumption can be reduced by moving* one *task from one processor to another. Otherwise, it is said to be* balanced.

It is clear that the power-optimal task-to-processor assignment must be balanced, since by definition its total energy consumption cannot be reduced. In the motivational example of Section 3, the first three task assignments are unbalanced, while the fourth(optimal) one is balanced. However, a balanced partitioning (as defined above) is not necessarily optimal. Consider the task assignment Π_1 which allocates four tasks to 2 processors as follows: $T_1(u_1 = 0.5)$ and $T_2(u_2 = 0.4)$ are assigned to M_1, while $T_3(u_3 = 0.4)$ and $T_4(u_4 = 0.3)$ are assigned to M_2. Π_1 is a balanced partitioning, but we can obtain another balanced (in fact, power-optimal) partitioning by *swapping* T_2 and T_4.

Nevertheless, the unbalanced/balanced partitioning concepts prove to be useful in understanding and evaluating the performance of partitioning heuristics: as we will see, some well-known heuristics from traditional (non power-aware) scheduling theory tend to produce feasible yet unbalanced task assignments, with poor energy performance. In addition, it will also allow us to establish that any task whose utilization exceeds a certain threshold must be allocated to a separate processor, exclusively. We can now formally characterize (un)balanced task assignments:

Proposition 1 *A task-to-processor assignment* Π *is unbalanced if and only if there exist two processors* M_i, M_j *and a task* T_a *assigned to* M_i *such that* $U_i(\Pi) - U_j(\Pi) > u_a$.

Proof: If part: Suppose that there exists a task assignment Π that contradicts the statement. Then there must be two processors M_i and M_j such that $U_i(\Pi) - U_j(\Pi) = K > 0$ and at least one task T_a assigned to M_i with $u_a < K$. Consider the new partitioning Π' obtained from Π by *transferring* T_a from M_i to M_j. Since the function $g()$ is strictly convex and $0 < u_a < K$, $g(U_i - u_a) + g(U_j + u_a) < g(U_i) + g(U_j)$, thus the total energy consumption of partitioning Π' given by (3) is definitely smaller. Further, $U_j + u_a < U_i \leq 1.0$ and the feasibility is preserved in the new partitioning.
Only if part: Suppose that the condition given in the proposition is not satisfied, yet it is possible to reduce the energy consumption by moving only one task T_a from M_i to M_j in partitioning Π. There are two possibilities:

i. $U_i - U_j = K > 0$. In this case, to be consistent with the assumptions, u_a must be equal to $K + D$ with $D > 0$. But, in the *new* partitioning, $U'_j - U'_i = K + 2D > K + D$. Hence, the new partitioning is unbalanced and by *moving back* T_a to M_i (and thereby returning to the original partitioning Π) we should be able to reduce energy consumption once again with respect to Π, clearly a contradiction.

ii. $U_j - U_i = K \geq 0$. That is, we are moving the task from the lightly loaded processor to the heavily loaded one. The resulting partitioning can be easily seen to be unbalanced, and just like the case of (i.) we should be able to further improve the energy savings by returning to the *original* partitioning; a contradiction.

□

In a partitioning Π, any pair of processors (M_i, M_j) for which the condition stated in Proposition 1 is satisfied is said to form an *unbalanced pair*. Now, consider the *average load per processor* defined as $A = \frac{\sum_{j=1}^{n} u_j}{m} = \frac{U_{tot}}{m}$. With the help of load balancing approach, we can prove the following property of power-optimal partitionings.

Theorem 2 *In power-optimal partitioning, a separate processor is assigned exclusively to each task* T_a *such that* $u_a > A$.

Proof: Suppose the contrary, that is, there exists a power-optimal partitioning where another task T_b is allocated to a machine M_j in addition to T_a where $u_a > A$. Clearly, $U_j > A + u_b$. Now there must be at least one processor M_i with load $U_i < A$ (otherwise the total load on all the processors would be at least $(m-1)A + A + u_j = mA + u_j > mA$). But if this is the case, the supposedly power-optimal partitioning is unbalanced ($U_j - U_i > u_b$ and T_b is assigned to U_j). Since an unbalanced partitioning cannot be power-optimal, we reach a contradiction.

□

5 Heuristics for POWER-PARTITION

A wealth of *efficient* heuristics are already available for the *feasibility* aspect of the problem from Multiprocessor Real-Time Scheduling: these include First-Fit (FF), Best-Fit (BF), Next-Fit(NF), Worst-Fit(WF), among others [5, 13, 4]. These algorithms process the tasks one by one, assigning each task to a processor according to the heuristic function that decides how to break ties if there are multiple processors that can accomodate the new task.

If the characteristics of the task set are available a-priori, then it is known that *ordering the tasks according to non-increasing utilizations* (or in bin-packing, ordering the items according to their sizes) improves the performance [5]. A recent and particularly important result for our investigation is due to Lopez [14]: Any *reasonable* task allocation algorithm which first orders tasks according to utilization values is optimal in the sense that the minimum achievable utilization bound of *no other* reasonable allocation algorithm can provide a better bound [14].

If the algorithm is allowed to preorder the task set according to utilizations, the term *decreasing* is added to its name [5]: for example, we have Best-Fit Decreasing (BFD) version of Best-Fit (BF) algorithm. Following [14], we call this class *Reasonable Allocation Decreasing* (or RAD, for short). Our simulation results support the expectation that the *average* case performance of this class of heuristics *improves* as well when they are first allowed to preorder tasks.

However, if tasks arrive dynamically and the scheduler is expected to assign each task to a processor without having information about the characteristics of tasks that may arrive in the future, then the decreasing version of heuristics cannot be applied. For this reason, we will also provide an analysis of the case where the scheduler is *not* allowed to re-order task set.

Besides feasibility, our problem has an equally important goal: minimizing the energy consumption. That is, we have to explore the energy consumption characteristics of each heuristic (in addition to feasibility performance). As we will see, when considered together, in some cases the *feasibility* and *energy* performances do not point to a "clear winner"; to deal with such scenarios, we propose an additional metric called *Timeliness/Energy* that combines performances in both dimensions.

Simulation Settings: We have generated a total of 1000000 task sets by varying the number of processors m, the total utilization of the task set $U = U_{tot}$ and the number of tasks n. In addition to these, the *individual task utilization factor* α [13] has been another key parameter: Having a task set with individual task utilization factor α means that the utilization of no *single* task exceeds α. Clearly, for a given task set, $\frac{U_{tot}}{n} \leq \alpha \leq 1.0$ must hold. We note that having no constraints (or information) about the individual task utilizations is equivalent to setting α to 1.0. When focusing on a multiprocessor platform with m processors, we modified U between $\frac{m}{10}$ (lightly loaded system) and m (heavily loaded system). For a given total utilization value U, we modified α between $\frac{U_{tot}}{n}$ and 1.0 to explore the effect of individual task utilization factor. We considered systems with 2, 4, 8, 16 and 32 processors while generating task sets with 50, 100 and 150 tasks. Due to lack of space, we present our results only in the context of 100-task sets that are to be scheduled on 16 processors, however we must underline that the trends and relative performances of techniques are similar in other settings as well.

For each heuristic H, we present:

- The *feasibility performance* (FP_H), given as the percentage of task sets that are feasibly scheduled by H.
- The *energy consumption performance* (EC_H), given as average energy consumption of task sets that are scheduled by H in feasible manner.
- The *timeliness/energy* metric, given as $\frac{FP_H}{EC_H}$.

Note that the last metric favors the heuristics with high feasibility performance and low energy consumption.

5.1 Performance of Algorithms with Utilization Ordering

By examining the performance of heuristics when they are allowed to order tasks according to utilization values, we observe that Worst-Fit-Decreasing is by far the best heuristic in terms of overall performance: Although its feasibility performance is not the best, it is comparable to other heuristics' performances even at high utilizations and high α values(Figures 3 and 4). However, its energy consumption performance clearly dominates all others, throughout the entire utilization and α spectrum (Figures 5 and 6). This fact is even more emphasized by the *timeliness/energy* curves of heuristics (Figures 7 and 8).

Albeit good in terms of feasibility performance, First-Fit-Decreasing and Best-Fit-Decreasing heuristics' performances suffer from *energy* point of view: These algorithms greedily schedule the tasks on one processor to the extent it is possible while keeping other processors idle, and this results in *unbalanced* partitionings in many cases. It is also interesting to note that FFD and BFD are hardly distinguishable in both energy and feasibility dimensions in this set of experiments.

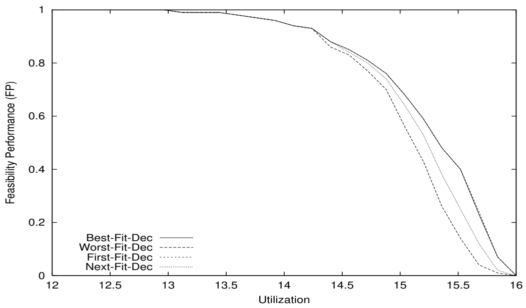

Figure 3. Feasibility Performance for $\alpha = 1.0$

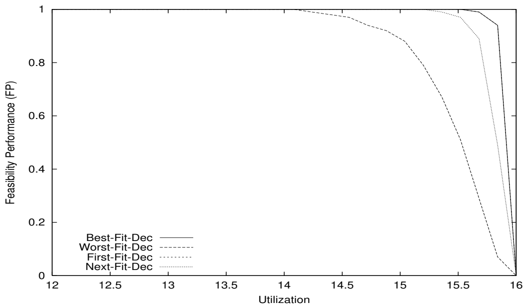

Figure 4. Feasibility Performance for $\alpha = 0.5$

In fact, it is possible to give a formal explanation of WFD's good performance through the following theorem.

Theorem 3 *Worst-Fit Decreasing (WFD) heuristic never produces an unbalanced partitioning.*

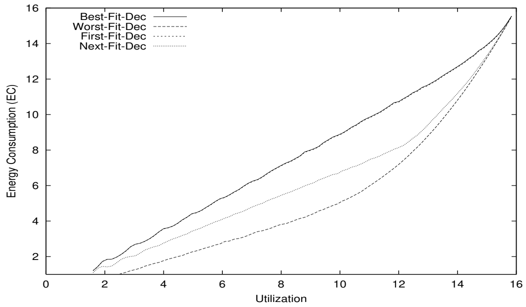

Figure 5. Energy Performance for $\alpha = 1.0$

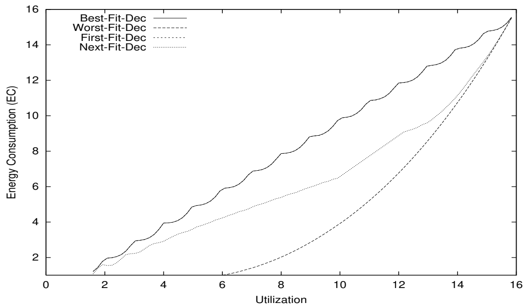

Figure 6. Energy Performance for $\alpha = 0.5$

Proof: Consider a set $\mathcal{T}$ of n periodic tasks (labeled according to non-increasing utilizations) that are to be scheduled on m processors. We will prove the statement by induction. Clearly, the partitioning after assigning the first task T_1 to an arbitrary idle processor is balanced. Suppose that the statement holds after assigning $T_1, \ldots, T_k (1 \leq k < n)$ to the processors according to WFD heuristic. Call the partitioning after assigning the k^{th} task Π_k. For convenience, processors are indexed according to non-increasing load values in Π_k in the following manner: M_1 is the processor with the highest load value, M_2 is the processor with second highest load, and so on.

WFD chooses M_m to allocate T_{k+1}. Observe that any pair (M_i, M_j) such that $i \neq m$ and $j \neq m$ cannot be the source of an unbalanced partitioning; because their loads did not change and Π_k is supposed to be balanced by induction assumption. Any pair (M_m, M_i) such that $U_i = 0$ cannot be unbalanced either: *Only* T_{k+1} must be assigned to M_m (otherwise WFD would not choose M_m) and such a pair is clearly balanced. So, we need to focus only on pairs (M_m, M_i) such that $U_i > 0$. We distinguish two cases:

i. After assignment of T_{k+1} to M_m, $U_m < U_i$ for each processor such that $U_i > 0$. In this case, the new assignment cannot result in an unbalanced "pair" (M_m, M_i), because if it were, then the same pair would be balanced in Π_k as well (we only reduced the difference between M_m and other processors with non-zero load).

ii. After assignment of T_{k+1} to M_m, $U_m \geq U_i$ for some processors M_i with non-zero load. We do not need the consider M_m and any other processor with *higher* load for potential "balance analysis" (the same reasoning as in i.) Consider a pair (M_m, M_i) such that $U_m \geq U_i > 0$ in Π_{k+1}. Observe that in Π_k, $U_m \leq U_i$ and thanks to the pre-ordering of tasks according to utilizations $u_{k+1} \leq U_i$. Furthermore, in Π_{k+1}, T_{k+1} must be the task with smallest utilization on M_m. Thus, after allocation, $U_m - U_i \leq u_{k+1}$ and in fact, $U_m - U_i \leq u_x$ for any task T_x allocated to M_m $(x \leq k+1)$. Under such conditions, the pair (M_m, M_i) cannot be unbalanced. □

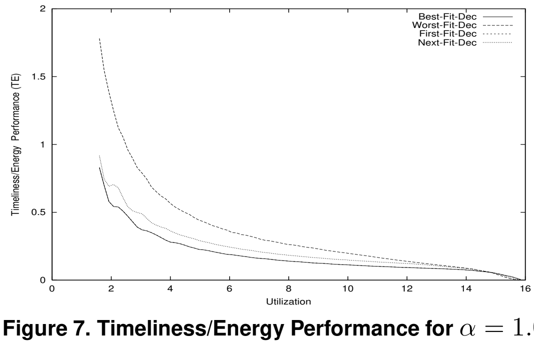

Figure 7. Timeliness/Energy Performance for $\alpha = 1.0$

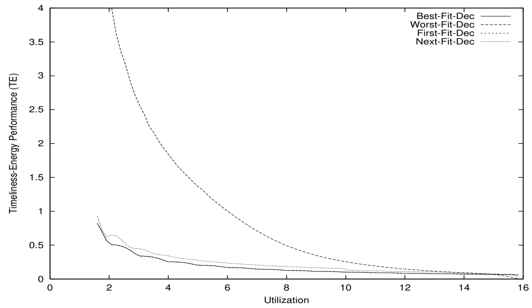

Figure 8. Timeliness/Energy Performance for $\alpha = 0.5$

It can be shown that the previously mentioned property of power-optimal partitioning in Theorem 2 holds in all partitionings produced by WFD:

Proposition 2 *WFD always generates a partitioning where a separate processor is exclusively assigned to any task T_a with utilization greater than* $A = \frac{\sum_{j=1}^{n} u_j}{m} = \frac{U_{tot}}{m}$.

Proof: Justified by the fact that partitionings where other tasks are allocated to the same processor as T_a would be necessarily unbalanced (see the proof of Theorem 2) and Theorem 3 (WFD never produces unbalanced partitionings). □

5.2 Performance of Algorithms without Utilization Ordering

If the scheduler algorithm does not have full information about individual tasks, then we will have to assign tasks as they are submitted to the system *without* being able to pre-order according to utilization values.

If we restrict our analysis to traditional heuristics First-Fit, Best-Fit, Next-Fit and Worst-Fit, we observe that we no longer have a clear winner that offers good performance in all utilization values in terms of both feasibility and energy consumption(Figures 9-12). FF, NF and BF all offer good performances in terms of feasibility, but their energy consumption characteristics are poor, especially at low utilizations. WF offers good performance at low utilizations, though its feasibility performance degrades rapidly with increasing utilization.

In fact, one can see that if WF is not allowed to order tasks according to utilizations before proceeding, its worst-case performance in terms of achievable utilization is extremely bad: Consider $m+1$ tasks that are to be executed on m processors, where $u_1 = u_2 = \ldots u_m = \epsilon$ and $u_{m+1} = 1.0$. $U_{tot} = 1 + m\epsilon$ (arbitrarily close to 1.0) and the total available computational capacity is m, yet WF produces an infeasible partitioning.

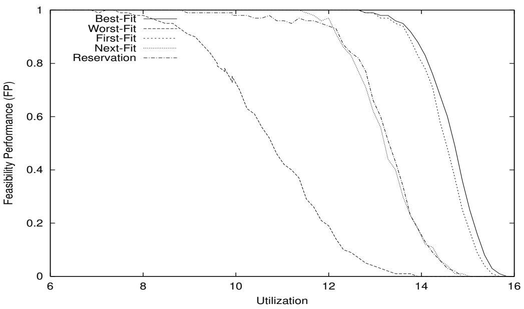

Figure 9. Feasibility Performance for $\alpha = 1.0$

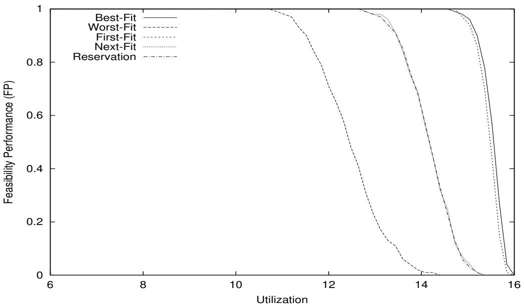

Figure 10. Feasibility Performance for $\alpha = 0.5$

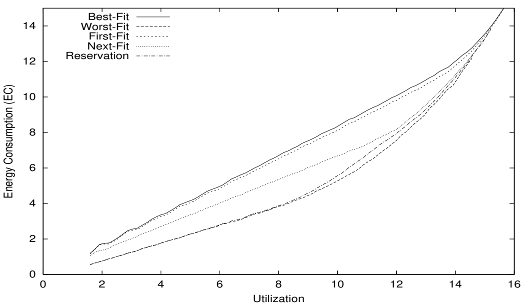

Figure 11. Energy Performance for $\alpha = 1.0$

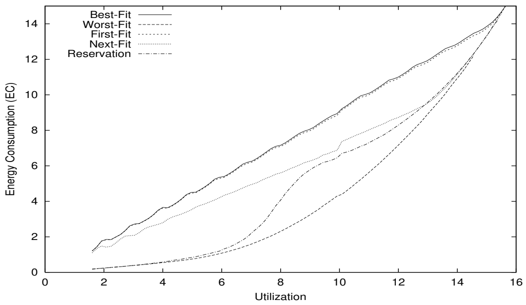

Figure 12. Energy Performance for $\alpha = 0.5$

To overcome these difficulties, we present an algorithm called RESERVATION. The idea of the algorithm is to *reserve* half (more accurately, $\lfloor m/2 \rfloor$ processors) of processor set for "light" tasks, and the other half for "heavy" tasks. A *light* task is defined to be a task with utilization not exceeding $A = \frac{U_{tot}}{m}$, average utilization per processor. Otherwise, the task is said to be *heavy*. When presented a task T_i, the algorithm tries to allocate it to the corresponding subset of processors (if there are multiple candidates in the corresponding subset, again Worst-Fit rule is used to break ties). Only when the corresponding subset is not able to accomodate the new task the *other* subset is tried (again, ties are broken using Worst-Fit).

RESERVATION algorithm is in fact a trade-off between the good feasibility performance of First/Best-Fit algorithms and the good energy performance of Worst-Fit algorithms. Figures 13 and 14 show that RESERVATION algorithm achieves a more or less *consistent* performance throughout the utilization spectrum. Further, its *Timeliness/Energy* performance is consistent with varying α parameter (Figure 15).

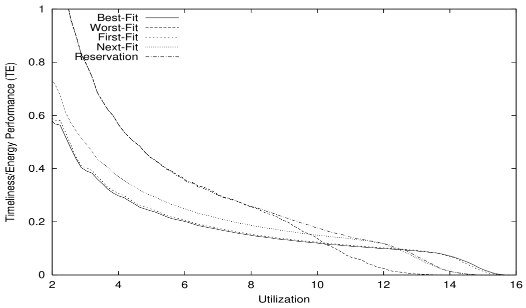

Figure 13. Timeliness/Energy Performance for $\alpha = 1.0$

6 Conclusion

To the best of our knowledge, this work is the first attempt to incorporate variable voltage scheduling of *periodic task sets* (hence, energy awareness issues) to *partitioned* multiprocessor real-time systems. We showed that finding the partitioning with minimum energy consumption is NP-Hard in the strong sense, even when the feasibility of the task set is guaranteed a priori. Then we developed our load balancing framework, showing that some partitionings are unbalanced in that moving

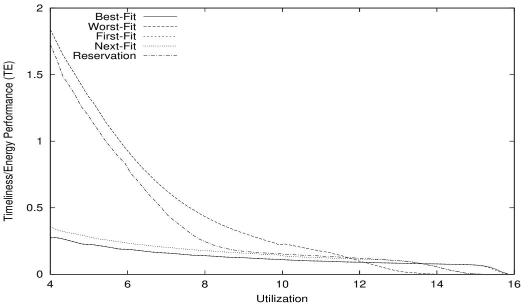

Figure 14. Timeliness/Energy Performance for $\alpha = 0.5$

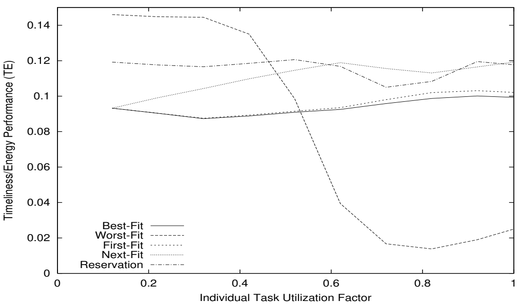

Figure 15. Effect of α on Timeliness/Energy performance for $U = 12$ on $m = 16$ processors

just one task from one processor to another can immediately improve energy savings. Our experimental evaluation shows that Worst-Fit-Decreasing heuristic is a clear winner in timeliness/energy performance. However, for the case where the algorithms are not allowed to preorder tasks according to utilizations, we proposed a new algorithm RESERVATION that does not exhibit large variances observed in other heuristics.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE International Real-Time Systems Symposium*, December 2001

[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 *Proceedings of the 13th EuroMicro Conference on Real-Time Systems (ECRTS'01)*, June 2001.

[3] H. Aydin, R. Melhem, D. Mossé and P.M. Alvarez. Dynamic and Aggressive Power-Aware Scheduling Techniques for Real-Time Systems. In *Proceedings of the 22nd IEEE Real-time Systems Symposium (RTSS'01)*, December 2001.

[4] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New strategies for Assigning Real-Time Tasks to Multiprocessor Systems. *IEEE Transactions on Computers*, 44(12), 1995.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation Algorithms for Bin Packing: A Survey. In *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, Boston (1997),

[6] S. Funk, J. Goossens and S. Baruah. Energy-minimization Techniques for Real-Time Scheduling on Multiprocessor platforms. *Technical Report 01-30*, Computer Science Department, University of North Carolina-Chapel Hill, 2001.

[7] J. Goossens, S. Baruah and S. Funk. Real-time Scheduling on Multiprocessors. *In Proceedings of the 10th International Conference on Real-Time Systems*, 2002.

[8] M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freman, NewYork, 1979.

[9] F. Gruian. System-Level Design Methods for Low-Energy Architectures Containing Variable Voltage Processors. In *Power-Aware Computing Systems Workshop at ASPLOS 2000*, 2000.

[10] I. Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, December 1998.

[11] S. Lauzac, R. Melhem and D. Mosse. An Efficient RMS Admission Control and its Application to Multiprocessor Scheduling. In *Proceedings of International Parallel Processing Symposium*, 1998.

[12] C.L. Liu and J.W.Layland. Scheduling Algorithms for Multiprogramming in Hard Real-time Environment. *Journal of ACM* 20(1), 1973.

[13] J. Lopez, J. Diaz, M. Garcia and D. Garcia. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proceedings of the 12th Euromicro Workshop on Real-Time Systems*, 2000.

[14] J. Lopez. Utilization Based Schedulability Analysis of Real-time systems Implemented on Multiprocessors with Partitioning Techniques. *Ph.D. Thesis*, University of Oviedo, 2001.

[15] W. Kim, D. Shin, H.S. Yun, J. Kim and S.L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proceedings of the 8th Real-Time and Embedded Technology and Applications Symposium*, 2002.

[16] D. Oh and T. P. Baker. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. *Real-Time Systems*, 15(2), 1998.

[17] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proceedings of the 36th Design Automation Conference, DAC'99*, pp. 134-139.

[18] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Kerkest and R. Lauwereins. Energy-Aware Runtime Scheduling for Embedded-Multiprocessor SOCs. In *IEEE Design and Test of Computers*, 18(5), 2001.

[19] F. Yao, A. Demers and S. Shankar. A Scheduling Model for Reduced CPU Energy. *IEEE Annual Foundations of Computer Science*, pp. 374 - 382, 1995.

[20] D. Zhu, R. Melhem, and B. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems. In *Proceedings of the 22nd IEEE Real-time Systems Symposium*, 2001.

[21] D. Zhu, N. AbouGhazaleh, D. Mosse and R. Melhem. Power Aware Scheduling for AND/OR Graphs in Multi-Processor Real-Time Systems. In *Proceedings of International Conference on Parallel Processing*, 2002.