# System-level Energy Management for Periodic Real-Time Tasks *

Hakan Aydin Vinay Devadas
Department of Computer Science
George Mason University
Fairfax, VA 22030
{aydin, vdevadas}@cs.gmu.edu

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
dzhu@cs.utsa.edu

## Abstract

In this paper, we consider the system-wide energy management problem for a set of periodic real-time tasks running on a DVS-enabled processor. Our solution uses a generalized power model, in which frequency-dependent and frequency-independent power components are explicitly considered. Further, variations in power dissipations and on-chip/off-chip access patterns of different tasks are encoded in the problem formulation. Using this generalized power model, we show that it is possible to obtain analytically the task-level energy-efficient speed below which DVS starts to affect overall energy consumption negatively. Then, we formulate the system-wide energy management problem as a non-linear optimization problem and provide a polynomial-time solution. We also provide a dynamic slack reclaiming extension which considers the effects of slow-down on the system-wide energy consumption. Our experimental evaluation shows that the optimal solution provides significant (up to 50%) gains over the previous solutions that focused on dynamic CPU power at the expense of ignoring other power components.

## 1 Introduction

With the advance of pervasive computing and on-going miniaturization of computers, the energy management has become a major research area in Computer Science and Engineering. A widely popular energy management technique, *Dynamic Voltage Scaling (DVS)*, is based on adjusting the CPU voltage and frequency on-the-fly [27]. Since the dynamic CPU power is a strictly increasing convex function of the CPU speed, the DVS techniques attempt to reduce the CPU speed to the extent it is possible, to obtain higher energy savings. The tasks' response times tend to increase with the reduced CPU speed: hence, for real-time systems where timeliness is crucial, special provisions must be taken to guarantee the timing constraints when DVS is applied. Consequently, the following *Real-Time DVS (RT-DVS)* problem attracted much attention in the research community: *Minimize the energy consumption through DVS while still meeting all the deadlines of the real-time tasks*. In the last decade, literally hundreds of research studies were published, each tackling the RT-DVS problem for a different system/task model and/or improving the existing solutions [2, 3, 19, 22, 20].

More recently, some research groups fine-tuned the problem, by observing that the task execution times do not always scale linearly with the CPU speed [5, 9, 23]: In fact, the off-chip access times (e.g. main memory or I/O device access latencies) are mostly independent of the CPU frequency. These works correctly observe that the task's workload has a *frequency-dependent (on-chip)* and another *frequency-independent (off-chip)* component. The latter does not scale with the CPU frequency, and it may be possible to further reduce the CPU speed beyond the levels suggested by the earlier studies [2, 19, 22], without compromising the feasibility. Note that an implicit assumption/motivation of this line of research is that further reduction of the CPU speed will bring additional energy savings.

Despite the depth and, in many cases, the analytical sophistication of these RT-DVS studies, there is a growing recognition that for more effective energy management, one should consider the *global* effects of DVS on the computer system. In particular, focusing exclusively on the *frequency-dependent CPU power* may hinder the system-level energy management: *the total energy consumption of the system depends on multiple system components (not only CPU)* and *the power dissipation does not comprise solely the frequency-dependent active power*. In fact, two recent experimental studies [10, 24] report that DVS *can increase* the total energy consumption; mainly because, with increased task execution times, it may force the components other than the CPU (e.g. memory, I/O devices) to remain in active state for longer time intervals.

The main objective of this research effort is to provide a generalized energy management framework for periodic real-time tasks. Specifically, when developing our solution, we simultaneously consider:

(a) A generalized power model which includes the *static*, *frequency-independent active* and *frequency-dependent active* power components of the *entire* system,

(b) Variations in the system power dissipation during the execution of *different* tasks, and

*This work was supported, in part, by US National Science Foundation CAREER award (CNS-0546244).

(c) On-chip / off-chip workload characteristics of individual tasks.

Given the above information, we show how to compute the *task-level energy-efficient speed* below which DVS starts to *adversely* affect the overall energy consumption of the system. We formulate the system-level energy management problem for periodic real-time tasks as a non-linear optimization problem. Then, we show how the task-level optimal speed assignments (to minimize the overall system energy) can be computed in polynomial time. Finally, we provide a dynamic reclaiming extension for settings where tasks may complete early, to boost energy savings at run-time. As required in real-time system design, our solutions assure that all the deadlines are met in both static and dynamic solutions. We also present simulation results showing the gains yielded by our optimal algorithm, for a wide range of system parameters.

We note that a number of research groups explored the issues beyond DVS-based CPU power management: for example static/leakage power management issues were investigated in [13, 14, 21]. The interplay between device power management and DVS are explored in [8, 16, 18, 25] from different aspects. Studies in [11, 31] also considered the problem of system-wide energy minimization for real-time task sets (under different power models), but they did not consider the effects of off-chip and on-chip workloads, and the proposed solutions were essentially heuristic-based. To the best of our knowledge, our work is the first research effort to provide a *provably optimal* and *analytical* solution to the *system-level* enery management problem for *periodic* real-time tasks, while keeping an eye on *all three* fundamental dimensions (a), (b) and (c) above.

# 2 System Model and Assumptions

## 2.1 Task and Processor Model

We consider a set of independent periodic real-time tasks $\Psi = \{\tau_1, \ldots, \tau_n\}$ that are be to executed on a uniprocessor system according to the preemptive Earliest-Deadline-First (EDF) policy. The period of task $\tau_i$ is denoted by $T_i$, and the relative deadline of each task instance (job) is equal to the task period. The $j^{th}$ instance of task $\tau_i$ is denoted by $\tau_{i,j}$. We assume a variable speed (DVS-enabled) processor whose speed (frequency) $S$ can vary between a lower bound $S_{min}$ 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 worst-case execution time $C(S)$ of task $\tau_i$ at CPU speed $S$ is given by $C_i(S) = \frac{x_i}{S} + y_i$ where $x_i$ is the task's on-chip workload at $S_{max}$, and $y_i$ is the task's off-chip workload (that does not scale with the CPU speed). Similarly, we define the task's *effective* utilization $U_i(S)$ at CPU speed $S$ as $\frac{u_i^x}{S} + u_i^y$. Here, $u_i^x = \frac{x_i}{T_i}$ and $u_i^y = \frac{y_i}{T_i}$ represent the utilization of the on-chip workload (at maximum speed) and off-chip workload of $\tau_i$, respectively.

The *base total utilization* $U_{tot}$ of the task set is the aggregate utilization of all the tasks at the maximum CPU speed, that is $U_{tot} = \sum_{i=1}^{n} U_i(S_{max})$. The necessary condition for the feasibility of the task set is $U_{tot} \leq 1.0$, further, this is also sufficient in the case of preemptive EDF scheduling policy [15]. Hence, throughout the paper, we will assume that $U_{tot}$ does not exceed 100%.

We assume that the process descriptor of each task/job is augmented by two extra fields, the *current speed* and the *nominal speed*. The former denotes the speed level at which the task is executing and the latter represents the "default" speed it has whenever it is dispatched by the operating system prior to any dynamic adjustment. The current and nominal speeds of the periodic task $\tau_i$ are denoted by $S_i$ and $\widehat{S}_i$, respectively. Note that the task set's total *effective* utilization on DVS-enabled settings depends on the speed assignments of individual tasks and is given by $\sum_{i=1}^{n} U_i(S_i)$. Although the results in this paper are derived assuming a continuous CPU speed spectrum, one can always "adapt" these solutions to discrete-speed settings by using a combination of two nearest available speed levels [12].

## 2.2 Power Model

In this paper, we adopt a generalized form of the system-level power model which was originally proposed in [30] and used in [28, 29]. In our general system-level power model, the power consumption $P$ is given by

$$P = P_s + \hbar(P_{ind,i} + P_{dep,i}) \quad (1)$$

where $P_s$ represents the *static power*, which may be removed only by powering off the whole system. $P_s$ includes (but not limited to) the power to maintain basic circuits, keep the clock running and the memory and I/O devices in sleep (or, standby) mode.

$P_{ind,i}$ and $P_{dep,i}$ represent the *frequency-independent active power* and *frequency-dependent active power* of the currently running task $\tau_i$, respectively. The frequency-independent power corresponds to the power component that does not vary with the system supply voltages and processing frequencies. Typically, the active power consumed by off-chip components such as main memory and/or I/O devices used by task $\tau_i$ would be included in $P_{ind,i}$. Note that the power of these devices can be efficiently removed only by switching to the sleep state(s) [10].

$P_{dep,i}$ includes the processor's dynamic power as well as any power that depends on system supply voltages and processing frequencies [7]. $P_{dep,i}$ depends on the CPU speed (clock frequency), as well as the effective switching capac-

itance $C_{f,i}$ of task $\tau_i$. $P_{dep,i}$ can be usually expressed as $C_{f,i} \cdot S_i^m$, where dynamic power exponent $m$ is a constant between 2 and 3, and $S_i$ is the current speed of $\tau_i$.

The coefficient $\hbar$ represents system states and whether active powers are currently consumed in the system. Specifically, $\hbar = 1$ if the system is *active* (defined as having computation in progress); otherwise (i.e. the system is in sleep mode or turned off) $\hbar = 0$.

Despite its simplicity, the above model captures the essential components of power consumption in real-time systems for system-level energy management. In fact, one contribution of this paper is to extend the model in [28, 30] to the cases where the frequency-independent and frequency-dependent active power figures can *vary* from task to task.

The time and energy overhead of completely turning off and turning on a device that is actively used by any task may be prohibitive [6]. Consequently, for the real-time embedded systems considered, we assume that several components may be put to low-power sleep states for energy savings but they are never turned off completely during the execution. In other words, $P_s$ is not manageable (i.e. it is always consumed). Hence, our target is to obtain energy savings through managing the frequency-independent and frequency-dependent *active* power discussion.

# 3 Derivation of Energy-Efficient Speed

Consider a real-time task $\tau$ with on-chip workload $x$ and off-chip workload $y$. The execution time af $\tau$ at the CPU frequency (speed) $S$ is given by $C(S) = \frac{x}{S} + y$. The sum of the *frequency-dependent* and *frequency-independent* components of the energy consumption of task $\tau$ at speed $S$ is:

$$\begin{aligned} E(S) &= (P_{dep}(S) + P_{ind}) \cdot (\frac{x}{S} + y) \\ &= P_{dep}(S) \cdot \frac{x}{S} + P_{dep}(S) \cdot y + P_{ind} \cdot \frac{x}{S} + P_{ind} \cdot y \end{aligned} \tag{2}$$

Recalling that $P_{dep}(S)$ is given as $C_f \cdot S^m$ where $m \geq 2$, and $x, y, P_{ind}$ are all positive constants, we can see that each of the four terms appearing in the latter sum is *strictly convex* for all $S \geq 0$. Since the sum of convex functions is also convex, the basic principles of convex optimization [17] allow us to deduce the following.

**Proposition 1** *The task energy consumption function $E(S)$ is a strictly convex function on the set of positive numbers. $E(S)$ has a single local minimum value and the speed that minimizes $E(S)$ can be found by setting its derivative $E'(S)$ to zero.*

**Definition 1** *The CPU speed that minimizes the energy consumption of task $\tau$ is called the* energy-efficient speed *of $\tau$ and denoted by $S_{eff}$.*

For the most common cases where the *frequency-dependent* power is $C_f S^m$ with $m = 2$ or $m = 3$, setting the derivative of $E(S)$ will give rise to the *cubic* or *quartic* equations that can be solved *analytically* [26]. As an example, if $m = 3$, we obtain a polynomial equation of fourth degree (a quartic equation):

$$3\,C_f\,y\,S^4 + 2\,C_f\,S^3\,x - P_{ind}\,x = 0 \tag{3}$$

If the ratio of the task's off-chip workload to the on-chip workload is $\alpha = \frac{y}{x}$, Equation (3) is equivalent to:

$$3\,C_f\,\alpha\,S^4 + 2\,C_f\,S^3 - P_{ind} = 0 \tag{4}$$

Through the *Descartes' Rule of Signs* [26], one can check that this equation has exactly one positive real root, which corresponds to $S_{eff}$. Hence, *the energy-efficient speed $S_{eff}$ is uniquely determined by $C_f, \alpha$ and $P_{ind}$*. Let us denote the root of Equation (4) by $S_{eff}(C_f, \alpha, P_{ind})$. Simple algebraic manipulation verifies the following properties:

- $S_{eff}(C_f, \alpha_1, P_{ind}) \leq S_{eff}(C_f, \alpha_2, P_{ind})$ if $\alpha_1 \geq \alpha_2$. In other words, the energy-efficient speed *decreases* with increasing off-chip workload ratio for the same effective switching capacitance and frequency-independent power values.
- $S_{eff}(C_{f_1}, \alpha, P_{ind}) \leq S_{eff}(C_{f_2}, \alpha, P_{ind})$ if $C_{f_1} \geq C_{f_2}$. Hence, the energy-efficient speed *decreases* with increasing effective switching capacitance for the same off-chip/on-chip workload ratio and frequency-independent power values.
- $S_{eff}(C_f, \alpha, P_{ind_1}) \geq S_{eff}(C_f, \alpha, P_{ind_2})$ if $P_{ind_1} \geq P_{ind_2}$. This shows that the energy-efficient speed *increases* with increasing frequency-independent power for the same effective switching capacitance and $\alpha$ values.

Note that this result is the generalization of the energy-efficient speed derivation in [30], where the authors did not consider the off-chip access time of tasks. In fact, by setting $y = 0$ in Equation (3), one can re-derive the result in [30].

As a concrete example, we investigate how the energy-efficient speed of a single task changes, as we vary the frequency-independent active power ($P_{ind}$) and the ratio of task's off-chip workload to on-chip workload ratio (see Figure 1). The task's effective switching capacitance is set to unity, and the frequency-dependent active power is given by $S^3$. Observe that, the figure verifies the trends mentioned above. It is interesting to note that $S_{eff}$ is very close to 0.375 even for $P_{ind} = 0.1$[1], and keeps increasing with larger $P_{ind}$. The rate of increase is even higher for cases where the portion of the *on-chip* workload grows. In fact, if $y = 0$ (the

[1]The differences between the energy-efficient speed values corresponding to different $\alpha$ values at $P_{ind} = 0.1$ are extremely small.

assumption of most early DVS studies), the increase in $S_{eff}$ is sharpest.

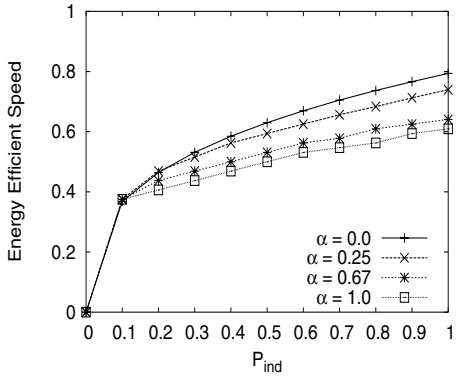


Figure 1: Energy-efficient speed as a function $P_{ind}$ and $\alpha$

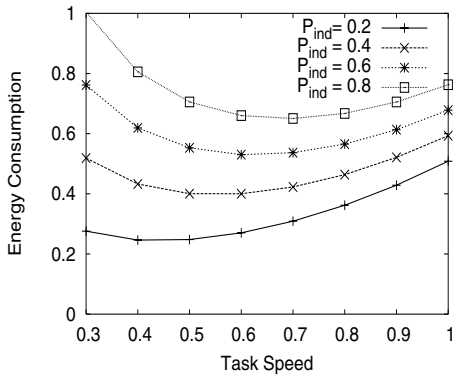


Figure 2: Energy consumption as a function of the CPU speed

Figure 2 depicts the effect of varying the CPU speed on the total energy consumption (for $\alpha = 0.25$). In these settings, $S_{eff}$ is never below $0.5$, and reducing the CPU speed below that level, in some cases, increases overall energy consumption even more significantly than using very high speeds. This example shows that violating the energy-efficient speed boundary may be detrimental for total energy consumption.

At this point, two observations are in order: First, since different tasks may use different I/O devices and may have different effective switching capacitance and off-chip/on-chip workload ratios, in general, the energy-efficient speed will vary from task to task. Second, the timing constraints of the task set may force the use of a higher speed for a given task $\tau_i$, but $S_i$ should never be reduced below $S_{eff,i}$ (which is $\tau_i$'s energy-efficient speed), because doing so increases the task (and system) energy consumption.

# 4 Static Optimal Solution

In this section, we consider the following problem: Given a DVS-enabled CPU and a periodic task set $\Psi = \{\tau_1, \ldots, \tau_n\}$, where each task may have different frequency-independent and frequency-dependent active power consumption characteristics as well as different on-chip execution and off-chip access times, find the task-level speed assignments that would minimize overall energy consumption while preserving the feasibility.

First, we would like to underline that it is relatively easy to justify the same speed assignment to different instances (jobs) of the same task: this follows from the convexity of task energy function[2].

Based on this, the energy minimization problem over the hyperperiod (least common multiple, LCM, of all the task periods) can be casted using the *utilization* information of tasks. Specifically, we can re-write Equation (2) as:

$$E_i(S_i) = (P_{dep,i}(S_i) + P_{ind,i}) \cdot (\frac{u_i^x}{S_i} + u_i^y) \cdot LCM \quad (5)$$

to express the energy consumption of task $\tau_i$ during the hyperperiod.

Then, the problem becomes finding the $\{S_i\}$ values so as to:

$$\text{minimize} \quad \sum_{i=1}^{n} E_i(S_i) \quad (6)$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{u_i^x}{S_i} \leq U_{bound} = 1 - \sum_{i=1}^{n} u_i^y \quad (7)$$

$$S_{min} \leq S_i \leq S_{max} \quad i = 1, \ldots, n \quad (8)$$

This is a *separable* convex optimization problem with convex constraints.

Above, the constraint (7) encodes the *feasibility* condition: with EDF, the effective utilization of the task set cannot exceed 100%. The effective utilization of the task set is given by $\sum_{i=1}^{n} \frac{u_i^x}{S_i} + \sum_{i=1}^{n} u_i^y$. Observe that the total utilization due to *off-chip* access times, namely $\sum_{i=1}^{n} u_i^y$, does not depend on the CPU speed. Hence, the total utilization due to the *on-chip computations*, namely, $\sum_{i=1}^{n} \frac{u_i^x}{S_i}$ is bounded by $U_{bound} = 1 - \sum_{i=1}^{n} u_i^y$. The constraint set (8) gives the minimum and maximum speed bounds of the processor.

As a first step, we process and modify the lower bound constraints as it is not beneficial to reduce the speed of any task below its energy-efficient speed. In fact, this also allows us to find the *optimal* speed values for some trivial cases.

**Proposition 2** *If $S_{eff,i} \geq S_{max}$ then $S_i = S_{max}$ in the optimal solution.*

This follows from the fact that $E_i(S_i)$ is convex on positive numbers: $E''(S_i) > 0$ and $E'(S_i)$ is strictly increasing. Since we know that $E_i'(S_{eff,i}) = 0$, we can see that $E_i'(S_i) < 0$ and $E_i(S_i)$ is strictly *decreasing* in the interval $(0, S_{eff,i}]$. If $S_{eff,i} \geq S_{max}$, choosing $S_i = S_{max}$ would minimize $E_i(S_i)$,

[2]If in the optimal solution multiple jobs had different speeds, assigning all the jobs a new and constant speed which is the *arithmetical mean* of existing assignments would hurt neither feasibility nor energy savings.

and consequently, the separable objective function. Further, running at the maximum speed can never have an adverse effect on the *feasibility*.

Hence, without loss of generality, in the remainder of the paper, we assume that $S_{eff,i} < S_{max}$ for all the tasks: if this does not hold for a given task $\tau_i$, we can easily set its speed to $S_i = S_{max}$, update $U_{bound}$ as $U_{bound} - u_i^x$ and obtain a smaller version of the problem with $n-1$ unknowns.

By defining $S_{low,i} = \max\{S_{min}, S_{eff,i}\}$, we can now incorporate the energy-efficient speed values to the formulation of the problem.

$$\text{minimize} \quad \sum_{i=1}^{n} E_i(S_i) \tag{9}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{u_i^x}{S_i} \leq U_{bound} = 1 - \sum_{i=1}^{n} u_i^y \tag{10}$$

$$S_{low,i} \leq S_i \leq S_{max} \quad i = 1, \ldots, n \tag{11}$$

We will distinguish two cases for the solution of the optimization problem above, depending on whether the quantity $\sum_{i=1}^{n} U_i(S_{low,i})$ (effective total utilization with $S_i = S_{low,i} \,\forall\, i$) exceeds 100% or not. Thanks to the transformation above, observe that $E_i(S_i)$ is strictly increasing in the interval $[S_{low,i}, S_{max}]$ for each task $\tau_i$. Hence, if total effective utilization does not exceed $100\%$ when all tasks run at their energy-efficient speeds (Case 1), then these speed assignments must be optimal:

**Proposition 3** *If $\sum_{i=1}^{n} U_i(S_{low,i}) \leq 1.0$ then $S_i = S_{low,i} \,\forall\, i$ in the optimal solution.*

Proposition 3 should be contrasted against the early results in RT-DVS research [2, 19], where the sole consideration of dynamic CPU power suggested reducing the speed as much as the feasibility allows. As can be seen, with frequency-independent power considerations, some CPU capacity may remain idle in the system-wide energy-optimal solution. For Case 2 ($\sum_{i=1}^{n} U_i(S_{low,i}) > 1.0$), we have the following.

**Proposition 4** *If $\sum_{i=1}^{n} U_i(S_{low,i}) > 1.0$ then, in the optimal solution, the total effective utilization $\sum_{i=1}^{n} U_i(S_i)$ is equal to 100 %.*

**Proof:** Suppose that $\sum_{i=1}^{n} U_i(S_{low,i}) > 1$, yet with optimal speed assignments $\{S_i\}$, $\sum_{i=1}^{n} U_i(S_i) < 1$. Let $\Delta = 1 - \sum_{i=1}^{n} U_i(S_i) > 0$. Note that since $\Delta > 0$, there must be at least one task $\tau_j$ that runs at a speed $S_j > S_{low,j}$. It is always possible to reduce $S_j$ by a small value $\epsilon$ in such a way that it remains *above* $S_{low,j}$ and that the total effective utilization is still below 100%. Since $E_i(S_i)$ is strictly increasing in the interval $[S_{low,i}, S_{max}]$, this would *reduce* the total energy consumption and still preserve the feasibility. We reach a contradiction since the proposed solution cannot be optimal.□

Thus, in the case where $\sum_{i=1}^{n} U_i(S_{low,i}) > 1.0$, we obtain the following optimization problem, denoted as problem ENERGY-LU.

$$\text{minimize} \quad \sum_{i=1}^{n} E_i(S_i) \tag{12}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{u_i^x}{S_i} = 1 - \sum_{i=1}^{n} u_i^y \tag{13}$$

$$S_{low,i} \leq S_i \quad i = 1, \ldots, n \tag{14}$$

$$S_i \leq S_{max} \quad i = 1, \ldots, n \tag{15}$$

This a separable convex optimization problem with $n$ unknowns, $2\,n$ inequality constraints and 1 equality constraint. This problem can be solved in iterative fashion in time $O(n^3)$, by carefully using the *Kuhn-Tucker* optimality conditions for convex optimization [17]. Full details of our solution are given in Appendix.

## 4.1 Experimental Results

In order to quantify the gains of the optimal scheme, we implemented a discrete event simulator. We generated 1000 synthetic task sets, each containing 20 periodic tasks. The periods of the tasks were chosen randomly in the interval [1000, 72000]. For each task set, we gradually increased the total utilization, and for each utilization value, we iteratively modified the proportion of the off-chip workload to the total (i.e. the summation of off-chip and on-chip) workload. For each data point, the energy consumption of the task set during the hyperperiod (LCM) is recorded and an average is computed. The effective capacitance and frequency-independent active power of each task was selected randomly according to uniform distribution in the interval $[0.1, 1.0]$. The frequency-dependent active power of task $\tau_i$ is given by $C_{f,i}\, S^3$. In this set of experiments, each task instance is assumed to present its worst-case workload.

We compare the performance of three schemes:

- Our *optimal* scheme, in which an optimal speed $S_i$ is computed separately for each task $\tau_i$ through the algorithm we described, considering various components of task's power consumption, before run-time.
- The scheme in which all tasks run with speed $S = U_{tot}$. Note that this speed is known to be optimal for periodic EDF scheduling [2, 19], *but when considering only the dynamic CPU power*.
- The scheme where all tasks run with the speed $S^* = \frac{\sum_{i=1}^{n} u_i^x}{1 - \sum_{i=1}^{n} u_i^y}$. This speed is known to be the *minimum speed* that still guarantees the feasibility, when one takes into account the information about the off-chip and on-chip workloads of tasks [5, 23]. In general, $S^* \leq U_{tot}$, hence, this approach enables the system to adopt even lower speed levels without compromising the feasibility.

Figure 3 presents the performance of the three schemes, as a function of the system utilization $U_{tot}$. The energy consumption figures are normalized with respect to the case where $U_{tot} = 100\%$. Here, the ratio of the off-chip workload to the total workload (namely, $\gamma = \frac{\sum_{i=1}^{n} u_i^y}{\sum_{i=1}^{n}(u_i^y + u_i^x)}$) is set to 0.2. The results show that when utilization is high (say, larger than 0.8), there is little difference between all three schemes, as the system has to use high CPU speed to guarantee feasibility. However, at low to medium utilization values, the advantage of using the optimal scheme becomes apparent (for $U_{tot} \leq 0.5$, gains around or exceeding $50\%$ are possible). It is also worth observing that the scheme using the speed $S^*$ performs *worse* than the scheme using $S = U_{tot}$, simply because, for many tasks, the *energy-efficient speed* was found to be *greater* than $S^*$.

In Figure 4, we show the effect of modifying the off-chip workload ratio $\gamma$, when $U_{tot}$ is fixed to 50%. The relative order of three schemes are the same, however, the advantage of using the optimal scheme over the (conventional) $S = U_{tot}$ approach becomes more emphasized at low off-chip workload ratios.

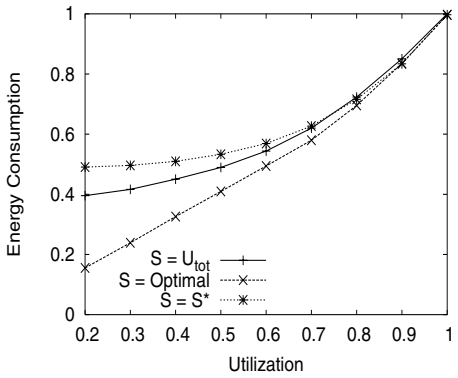


Figure 3: Energy consumption as a function of utilization

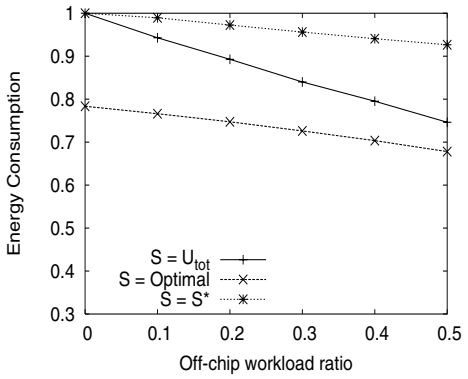


Figure 4: Energy consumption as a function of off-chip workload ratio

# 5 Dynamic Reclaiming for System-Level Power Management

The algorithm presented in Section 4 computes the optimal speed assignments to minimize the system-level total energy consumption, by taking into account task characteristics as well as frequency-dependent and independent active powers. However, in practice, even when one schedules all the tasks with the corresponding static optimal speeds, many task instances complete without presenting their worst-case workload in practice. In fact, *reclaiming* unused computation time to reduce the CPU speed while preserving feasibility was subject to numerous research papers in recent years [2, 3, 9, 19], though most of them focused exclusively on dynamic CPU power. In [2, 3], a generic dynamic reclamation algorithm (GDRA) was proposed for power-aware scheduling of *periodic* tasks. Later, the same algorithm was modified for use with power-aware scheduling of mixed real-time workloads (Extended Dynamic Reclaiming Algorithm - EDRA) [4] and energy-constrained scheduling of periodic tasks [1].

The algorithm we are about to present *(System-Level Dynamic Reclaiming Algorithm - SDRA)* is a generalization of EDRA for system-wide power management. Before presenting the specific enhancements, we provide a brief review of EDRA. In EDRA, each task instance $\tau_{i,j}$ assumed a **nominal (default)** speed of $\widehat{S}_i$. At dispatch time, this nominal speed was reduced by computing the unused CPU time of already-completed tasks (called the *earliness* factor of $\tau_{i,j}$).

EDRA is based on *dynamically* comparing the actual schedule to the static schedule $\mathcal{S}^{can}$(in which each task instance runs with its nominal speed and presents its worst-case workload). To perform the comparison, a data structure (called $\alpha$**-queue)** is maintained and updated at run-time. The $\alpha$-queue represents the ready queue of $\mathcal{S}^{can}$ at time $t$. Specifically, at any time $t$, the information about each task instance $\tau_{i,j}$ that *would* be ready at $t$ in $\mathcal{S}^{can}$is available in $\alpha$-queue, including its identity, ready time, deadline and *remaining execution time* (denoted by $rem_{i,j}$)[3]. EDRA assumes that tasks are scheduled according to EDF* policy [2]. EDF* is the same as EDF [15], except that, it provides a total order of all task priorities by breaking the ties in a specific way among task instances with the same deadlines [2]. The $\alpha$-queue is also ordered according to EDF* priorities. We denote the EDF* priority-level of task $\tau_i$ by $d_i^*$ (low values denote high priorities).

The key notation and rules pertaining to SDRA are presented in Figures 5 and 6, respectively. Rules 1-3 provide the update rules for the $\alpha$-queue structure at important "events" (task arrival/completion), while Rule 4 shows how the speed of a task $\tau_i$ is reduced by evaluating its earliness at dispatch

[3]Since there can be at most *one* ready instance of a periodic task at anytime $t$, we will drop the second index in $rem_{i,j}$ and in other $\alpha-$queue related notation.

time. $\epsilon_i(t)$ represents the unused computation time of tasks at higher or equal priority level with respect to $\tau_i$ at time $t$: these tasks would still have some non-zero remaining execution time in $\mathcal{S}^{can}$, but now they must have been completed because $\tau_i$ is the highest priority task in the system. Observe that $rem_i$ values are available in the $\alpha$-queue.

The difference between SDRA and EDRA [4] lies on the rules 4.3 and 4.4 and is justified on the following two principles:

(a.) For system-level energy-efficiency, a task's speed should never be reduced beyond $S_{low,i}$, even when the current earliness suggests doing so, and,

(b.) When the additional CPU time $\beta_i$ to be given to $\tau_i$ is determined, the new speed $S_i$ should be determined by taking into account both off-chip and on-chip components of the remaining workload.

Specifically, to achieve (a.) above, Rule 4.3 takes into account the difference between the remaining worst-case execution times of the task under the speed $S_{low,i}$ and current speed $S_i$, and compares against current earliness. Once the extra CPU time $\beta_i$ to be allocated to $\tau_i$ is determined, the algorithm computes the new speed $S_{new,i}$ by making sure that the equality $w_i^{S_i} + \beta_i = \frac{\bar{x}_i(t)}{S_{new,i}} + \bar{y}_i$ holds after the speed adjustment.

It is relatively easy to justify the correctness of SDRA, by taking into account that the additional CPU time reclaimed by SDRA is always smaller than or equal to the task's earliness (which means the algorithm is less aggressive than EDRA whose correctness was proven in [4]). Note that the speed computation at step 4.4 is determined uniquely by the extra CPU time allocation (which also determines the feasibility), and simply reflects the fact that the algorithm is cognizant of the off-chip/on-chip workload information of the running task.

## 5.1 Experimental Results

In this section, we experimentally evaluate the performance improvements due to dynamic reclaiming, and in particular, SDRA. The basic experimental settings are identical to those given in Section 4.1. However, one important addition is the fact that we also investigated the effect of variability in the *actual workload*. Specifically, for each (worst-case) utilization and off-chip/on-chip workload ratio, we simulated the execution of each task set 20 times over LCM, determining the *actual* execution time of each task instance (job) randomly at every release time. The actual workload of the task is determined by modifying the $\frac{BCET}{WCET}$ ratio (that is, the best-case to worst-case execution time ratio). The actual execution time of each job is chosen randomly, following a uniform probability distribution in the interval $[BCET, WCET]$.

- $\mathcal{S}^{can}$: The "canonical" schedule in which each task $\tau_i$ runs with its nominal speed $\widehat{S}_i$ and presents its worst-case workload $C_i = \frac{x_i}{\widehat{S}_i} + y_i$ at every instance
- $rem_i(t)$: the remaining execution time of $\tau_i$ at time $t$ in $\mathcal{S}^{can}$
- $\bar{x}_i(t)$: the remaining on-chip workload of task $\tau_i$ in the actual schedule at time $t$
- $\bar{y}_i(t)$: the remaining off-chip workload of task $\tau_i$ in the actual schedule at time $t$
- $w_i^S(t)$: the remaining worst-case execution time of task $\tau_i$ under the speed $S$ at time $t$ in the actual schedule, given by: $w_i^S(t) = \frac{\bar{x}_i(t)}{S} + \bar{y}_i$
- $\epsilon_i(t)$: The earliness of task $\tau_i$ at time $t$ in the actual schedule, defined as: $\epsilon_i(t) = \sum_{j|d_j^* < d_i^*} rem_j(t) + rem_i(t) - w_i^{\widehat{S}_i}(t) = \sum_{j|d_j^* \le d_i^*} rem_j(t) - w_i^{\widehat{S}_i}(t)$

Figure 5: Key Notation for SDRA

Rules for SDRA

1. Initialize the $\alpha$-queue to the empty-list.
2. At every event (arrival/completion), consider the head of the $\alpha$-queue and decrease its $rem_i$ value by the amount of elapsed-time since the last event. If $rem_i$ is smaller than the time elapsed since the last event, remove the head, update the time elapsed since the last event, and repeat the update with the next element. This is done until all "elapsed time" is used.
3. At every new arrival, insert into the $\alpha$-queue, in the correct EDF* order, the information regarding the new instance of task $\tau_{i,j}$ with $rem_i(t) = w_i^{\widehat{S}_i}$.
4. Whenever $\tau_i$ is about to be dispatched at time $t$:
   4.1. Set $S_i = \widehat{S}_i$.
   4.2. Consult the $\alpha$-queue and compute the earliness $\epsilon_i(t)$ of $\tau_i$.
   4.3. Compute the extra CPU time that will be given to $\tau_i$ as $\beta_i = \min\{\epsilon_i(t), w_i^{S_{low,i}} - w_i^{S_i}\}$.
   4.4. Reduce the speed of task $\tau_i$ by allocating an extra $\beta_i$ time units: $S_i = \frac{\bar{x}_i}{w_i^{S_i} + \beta_i - \bar{y}_i}$
5. At every event of preemption or completion of a task, say $\tau_i$, decrease the value of the remaining execution time: $w_i^{S_i} = w_i^{S_i} - \Delta_t$, where $\Delta_t$ is the time elapsed since the task $\tau_i$ was last dispatched.

Figure 6: System-Level Dynamic Reclaiming Algorithm (SDRA)

Figure 7 shows the performance comparison of three schemes with reclaiming enabled, when $\frac{BCET}{WCET} = 0.25$ and $\gamma = 0.2$. As tasks complete early, more energy savings can be obtained when compared to Figure 3 in all three

schemes, through reclaiming, and the optimal scheme's advantage is now emphasized even at high utilization values: when tasks complete early, SDRA re-uses the dynamic slack optimally, by considering energy-efficient speed threshold and task power/workload characteristics. Figure 8 presents the effects of varying $\gamma$ for the same settings, where $\frac{BCET}{WCET} = 0.25$ and $U_{tot} = 0.5$. Though energy gains are still more significant (due to reclaiming), it is interesting to note that the performance of $S = U_{tot}$ is highly sensitive to the the off-chip/on-chip workload characteristics. Finally, in Figure 9, the effects of varying the $\frac{BCET}{WCET}$ ratio are explored, for $U_{tot} = 0.5$ and $\gamma = 0.2$. The energy gains change almost linearly for all three schemes with varying $\frac{BCET}{WCET}$ ratio, and the gains of the optimal scheme are more pronounced when the variability in the workload is highest.

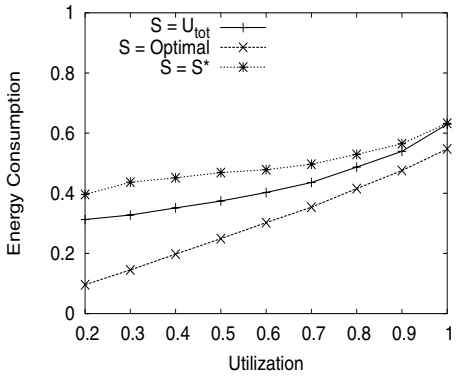


Figure 7: Energy consumption as a function of utilization (with dynamic reclaiming)

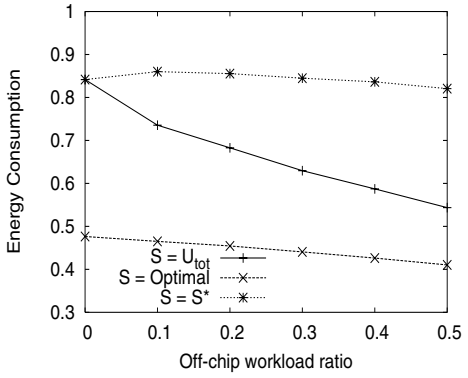


Figure 8: Energy consumption as a function of off-chip workload ratio (with dynamic reclaiming)

# 6 Conclusions

In this paper, we addressed the problem of minimizing overall energy consumption of a real-time system, considering a generalized power model. Unlike many previous RT-DVS studies in which the focus was dynamic CPU power, our solution takes into account the frequency-dependent and -independent power components, as well as the power consumption of components

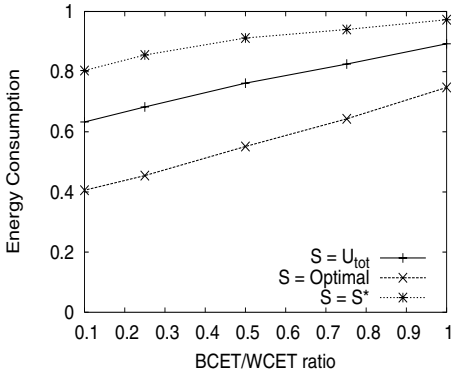


Figure 9: Energy consumption as a function of workload variability

other than the CPU. We also considered the off-chip/on-chip workload information of different tasks. After showing how the task-level energy-efficient speed levels can be computed in this generalized model, we formulated the problem as a convex optimization problem and derived an iterative, polynomial-time solution using Kuhn-Tucker optimality conditions. Finally, we provided a dynamic reclaiming extension for settings where tasks complete early. Our experimental results show that the gains of our optimal scheme are considerable, especially at low-to-medium utilization, or high workload variability conditions. To the best of our knowledge, this paper is the first research effort to provide an optimal solution for *system-level energy minimization* in real-time systems with periodic tasks, under a generalized power model.

# References


[1] T. A. AlEnawy and H. Aydin. Energy-Constrained Scheduling for Weakly Hard Real Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'05),* 2005.

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

[3] H. Aydin, R. Melhem, D. Mossé and P. Mejia-Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584-600, May 2004.

[4] H. Aydin and Q. Yang. Energy-Responsiveness Tradeoffs for Real-time Systems with Mixed Workload. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, 2004.

[5] E. Bini, G.C. Buttazzo and G. Lipari. Speed Modulation in Energy-Aware Real-Time Systems. in *Proc. of the* $17^{th}$ *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005.

[6] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. Mc- Dowell, and R. Rajamony. The case for power management in web servers, chapter 1. In *Power Aware Computing*. Plenum/Kluwer Publishers, 2002.

[7] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of The HICSS Conference*, 1995.

[8] H. Cheng and S. Goddard. Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation. In *Proc. of the* $2^{nd}$ *Int'l Workshop on Power-Aware Real-Time Computing (PARC)*, 2005.

[9] K. Choi, R. Soma and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proceedings of Design, Automation and Test in Europe,* 2004.

[10] X. Fan, C. Ellis, and A. Lebeck. The Synergy Between Power-aware Memory systems and Processor Voltage. In *Workshop on Power-Aware Computing Systems*, December 2003.

[11] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[12] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of The 1998 International Symposium on Low Power Electronics and Design*, 1998.

[13] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems. In *Proceedings of the 41st Annual Conference on Design Automation (DAC'04),* 2004.

[14] Y. Lee, K. P. Reddy, and C. M. Krishna. Scheduling Techniques for Reducing Leakage Power in Hard Real-time Systems. In *EuroMicro Conference on Real Time Systems (ECRTS'03)*, 2003.

[15] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-time Environment. *Journal of ACM* vol. 20. no. 1, pp. 46-61, January 1973.

[16] Y. Lu, L. Benini and G. De Micheli. Power-Aware Operating Systems for Interactive Systems.*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v.10, n.1, April 2002.

[17] D. Luenberger, *Linear and Nonlinear Programming,* Addison-Wesley, Reading Massachusetts, 1984.

[18] R. Nathuji and K. Schwan. Reducing System Level Power Consumption for Mobile and Embedded Platforms. In *Lecture Notes in Computer Science,* v. 3432, pp. 18-32, March 2005.

[19] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2001.

[20] A. Qadi, S. Goddard, and S. Farritor. A Dynamic Voltage Scaling Algorithm for Sporadic Tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'03),* 2003.

[21] G. Quan, L. Niu, X. S. Hu and B. Mochock. Fixed Priority Scheduling for Reducing Overall Energy on Variable Voltage Processors, In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'04)*, 2004.

[22] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-time Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, 2003.

[23] K. Seth, A. Anantaraman, F. Mueller and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. In *Proc. of the* $24^{th}$ *IEEE Real-Time System Symposium*, 2003.

[24] D. Snowdon, S. Ruocco and G. Heiser. Power Management and Dynamic Voltage Scaling: Myths and Facts. In *Proc. of the* $2^{nd}$ *Int'l Workshop on Power-Aware Real-Time Computing (PARC)*, 2005.

[25] V. Swaminathan and K. Chakrabarty. Energy-conscious, Deterministic I/O Device Scheduling in Hard Real-time Systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no.7, pp. 847-858, July 2003.

[26] H. W. Turnbull. *Theory of Equations.* Oliver and Boyd, London, 1947.

[27] M. Weiser, B. Welch, A. Demers and S. Shenker. Scheduling for Reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[28] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[29] D. Zhu and H. Aydin. Energy Management for Real-time Embedded Systems with Reliability Requirements. In *Proc. of the International Conference on Computer-Aided Design (ICCAD)*, 2006.

[30] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the International Conference on Computer Aided Design (ICCAD)*, 2004.

[31] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd Annual Conference on Design Automation (DAC)*, 2005.

# APPENDIX: Solution of Problem ENERGY-LU

Our strategy while solving the problem ENERGY-LU will be as follows. We will temporarily ignore the upper bound constraints (15), and obtain a new problem (called ENERGY-L) by considering only the inequality constraint (13) and the lower bound constraints (14). If the solution of ENERGY-L satisfies also the upper bound constraints (15), then it corresponds to the solution of ENERGY-LU, since the feasible region of the latter is contained in that of the former. However, if some upper bound constraints are violated, then we will show how to iteratively adjust the solution to solve ENERGY-LU. We first formally define the problem ENERGY-L:

$$\text{minimize } \sum_{i=1}^{n} E_i(S_i) \tag{16}$$

$$\text{subject to } \sum_{i=1}^{n} \frac{u_i^x}{S_i} = 1 - \sum_{i=1}^{n} u_i^y \tag{17}$$

$$S_{low,i} \leq S_i \quad i = 1, \ldots, n \tag{18}$$

Let $\Gamma = \{i | \frac{S_{max}^2}{u_i^x} \cdot E_i'(S_{max}) \leq \frac{S_{max}^2}{u_j^x} E_j'(S_{max}) \ \forall j\}$. Informally, $\Gamma$ contains the indices of tasks for which the quantity $\frac{S_{max}^2}{u_i^x} \cdot E_i'(S_{max})$ is minimum among all tasks.

**Lemma 1** *If the solution of ENERGY-L violates upper bound constraints given by (15) then, in the solution of ENERGY-LU,* $S_i = S_{max}$ $\forall i \in \Gamma$.

**Proof:** A well-known result of the nonlinear optimization theory states that the solution of Problem ENERGY-LU should satisfy so-called *Kuhn-Tucker conditions* [17]. Furthermore, Kuhn-Tucker conditions are also sufficient in the case of convex objective functions [17]. For the convex optimization Problem ENERGY-LU, Kuhn-Tucker conditions comprise the constraints (13), (14), (15) and :

$$E_i'(S_i) - \lambda \frac{u_i^x}{S_i^2} + \bar{\mu}_i - \mu_i = 0 \quad i = 1, 2, .., n \quad (19)$$

$$\bar{\mu}_i(S_{max} - S_i) = 0 \quad i = 1, 2, ., .n \quad (20)$$

$$\mu_i\,(S_i - S_{low,i}) = 0 \quad i = 1, 2, ., .n \quad (21)$$

$$\bar{\mu}_i \geq 0 \quad i = 1, 2, .., n \quad (22)$$

$$\mu_i \geq 0 \quad i = 1, 2, .., n \quad (23)$$

where $\lambda, \bar{\mu}_1, \bar{\mu}_2, \ldots, \bar{\mu}_n, \mu_1, \mu_2, \ldots, \mu_n$ are Lagrange multipliers.

The necessary and sufficient character of Kuhn-Tucker conditions indicates that any $3n + 1$ tuple $(S_1, \ldots, S_n, \mu_1, \ldots, \mu_n, \bar{\mu}_1, \ldots, \bar{\mu}_n, \lambda)$ which satisfies the conditions (19) ...(23) and constraints (13), (14), (15) provides optimal $S_i$ values for ENERGY-L.

Similarly, the necessary and sufficient Kuhn-Tucker conditions for the problem ENERGY-L include the constraints (17), (18), and :

$$E_i'(S_i) - \lambda \frac{u_i^x}{S_i^2} - \mu_i = 0 \quad i = 1, 2, .., n \quad (24)$$

$$\mu_i\,(S_i - S_{low,i}) = 0 \quad i = 1, 2, ., .n \quad (25)$$

$$\mu_i \geq 0 \quad i = 1, 2, .., n \quad (26)$$

where $\lambda, \mu_2, \ldots, \mu_n$ are again Lagrange multipliers.

Comparing the Kuhn-Tucker conditions for problems ENERGY-LU and ENERGY-L, we note that at least one $\bar{\mu}_i$ should be definitely larger than zero, if the solution of ENERGY-L violates some upper bound constraints of ENERGY-LU. This is because, if $\bar{\mu}_i = 0\,\forall\, i$, then in the Kuhn-Tucker conditions for problem ENERGY-LU, the equations (20) and (22) vanish, and the equation (19) becomes identical to the equation (24); implying that both sets of Kuhn-Tucker conditions (hence, the optimal solutions of two problems) coincide. But this contradicts the assumption that we made.

Similarly, if $\bar{\mu}_i > 0\ \exists\, i$, (which should be the case if the solution of ENERGY-L violates the constraints of ENERGY-LU), then $\mu_i$ should be zero: otherwise, Equations (20) and (21) would imply that $S_i = S_{max} = S_{low,i}$, which is a contradiction.

Now we are ready to prove the lemma. We will essentially show that, under the condition specified in the lemma, the Lagrange multipliers $\bar{\mu}_i, (i \in \Gamma)$ are all non-zero, which will imply (by (20)) that $S_i = S_{max}\ \forall\, i \in \Gamma$. Suppose $\exists m \in \Gamma$ such that $\bar{\mu}_m = 0$. From the discussion in the preceding paragraphs, we know that $\exists j \in \Gamma$ such that $\bar{\mu}_j > 0$ and $\mu_j = 0$. Using (19), we can write

$$\frac{S_m^2}{u_m^x} E_m'(S_m) - \frac{S_m^2}{u_m^x}\mu_m = \frac{S_j^2}{u_j^x} E_j'(S_j) + \frac{S_j^2}{u_j^x}\bar{\mu}_j$$

which gives (since $S_j = S_{max}$):

$$\frac{S_m^2}{u_m^x} E_m'(S_m) = \frac{S_{max}^2}{u_j^x} E_j'(S_{max}) + \frac{S_m^2}{u_m^x}\mu_m + \frac{S_{max}^2}{u_j^x}\bar{\mu}_j \quad (\mu_m \geq 0, \bar{\mu}_j \geq 0) \quad (27)$$

Since $S_m$ is necessarily less than or equal to $S_{max}$, we can conclude that $\frac{S_{max}^2}{u_j^x} E_j'(S_{max}) < \frac{S_m^2}{u_m^x} E_m'(S_m) \leq \frac{S_{max}^2}{u_m^x} E_m'(S_{max})$, which contradicts the assumption that $m \in \Gamma$. □

## Solving ENERGY-L

Having shown how to solve the problem ENERGY-LU if we have the solution of ENERGY-L, we now address the latter. First, we state a sufficient condition for the optimal solution of ENERGY-L.

**Lemma 2** *A set of speed assignments $S_X = \{S_1, \ldots, S_n\}$ is the solution to ENERGY-L, if it satisfies the property*

$$\frac{S_i^2}{u_i^x} E_i'(S_i) = \frac{S_j^2}{u_j^x} E_j'(S_j)\ \forall\, i, j \text{ such that } S_i, S_j \in S_X \quad (28)$$

*in addition to the constraint sets (17) and (18).*

**Proof:** By re-visiting the Kuhn-Tucker conditions (24), (25), (26), which are necessary and sufficient for optimality, we observe that the condition given in the lemma coincides with the case where $\mu_i = 0\ \forall i$. Indeed, in that case, the equation (24) becomes identical to (28), and all constraints involving $\mu_i$ variables vanish. If the $S_i$ values obtained in this way satisfy the constraint sets (18) and (17) at the same time, then the selection of $\mu_i = 0\ \forall i$ should be optimal as all Kuhn-Tucker conditions hold. □

Let us define by $h_i(\lambda)$ the inverse function of $\frac{S_i^2}{u_i^x} E_i'(S_i)$; in other words, $h_i(\lambda) = S_i$ if $\frac{S_i^2}{u_i^x} E_i'(S_i) = \lambda$. Note that, evaluating $h_i(\lambda)$ will typically involve solving a cubic or quadratic equation, for which closed form solutions do exist. Also, if we set a given $S_i$ value to a constant, then all other $S_j$ values are uniquely determined by the $h_i()$ functions. Moreover, $\frac{S_i^2}{u_i^x} E_i'(S_i)$ is strictly increasing with $S_i\ \forall\, i$. Hence, the unique set of speed assignments $S_X$ that satisfy (28) and (17), can be obtained by finding the unique $\lambda$ such that $\sum_{i=1}^{n} \frac{u_i^x}{h_i(\lambda)} = 1 - \sum_{i=1}^{n} u_i^y$. If $S_X$ satisfies also the lower bound constraints, then it is the solution of ENERGY-L, by virtue of Lemma 2. However, we need to deal with a last case, in which $S_X$ violates the lower bound constraints. In order to address this, we need to define a (last) set.

Let $\Pi = \{i | \frac{S_{low,i}^2}{u_i^x} E_i'(S_{low,i}) \geq \frac{S_{low,j}^2}{u_j^x} E_j'(S_{low,j})\ \ \forall j\}$. Informally, $\Pi$ contains the indices of tasks for which the quantity $\frac{S_{low,i}^2}{u_i^x} \cdot E_i'(S_{low,i})$ is maximum among all tasks.

**Lemma 3** *If $S_X$ violates the upper bound constraints given by (18), then, in the solution of ENERGY-L, $S_i = S_{low,i}\ \forall i \in \Pi$.*

**Proof:** The proof of Lemma 3 is similar to that of Lemma 1. In fact, if $S_X$ (which corresponds the the candidate solution when all $\mu_i$ values set to zero) violates the lower bound constraints, then $\exists j\, \mu_j > 0$ and $S_j = S_{low,j}$ (due to (25)). We will prove that, under the condition specified in the lemma, the Lagrange multipliers $\mu_i, i \in \Pi$ should be all non-zero, which will imply (by 25) that $S_i = S_{low,i} \forall\, i \in \Pi$.

Suppose $\exists m \in \Pi$ such that $\mu_m = 0$. From the preceding discussion, we know that $\exists j \in \Pi$ such that $\mu_j > 0$. Using (24), we can write $\frac{S_m^2}{u_m^x} E_m'(S_m) = \frac{S_j^2}{u_j^x} E_j'(S_j) - \frac{S_j^2}{u_j^x}\mu_j$, which gives (since $S_j = S_{low,j}$):

$$\frac{S_m^2}{u_m^x} E_m'(S_m) = \frac{S_{low,j}^2}{u_j^x} E_j'(S_{low,j}) - \frac{S_{low,j}^2}{u_j^x}\mu_j (\mu_j \geq 0) \quad (29)$$

At this point, since $S_m \geq S_{low,m}$, we can conclude that $\frac{S_{low,j}^2}{u_j^x} E_j'(S_{low,j}) > \frac{S_m^2}{u_m^x} E_m'(S_m) \geq \frac{S_{low,m}^2}{u_m^x} E_m'(S_{low,m})$, which contradicts the assumption that $m \in \Pi$.

**Complexity:** Note that ENERGY-L can be solved in time $O(n^2)$: evaluating $S_X$ takes linear time, and if $S_X$ violates (some) lower bound constraints, then we determine at least one optimal $S_i$ value at each iteration. If $S_X$ satisfies all the lower bound constraints, then the solution of ENERGY-L (at that iteration) coincides with $S_X$.
The same observation holds for ENERGY-LU: The algorithm will invoke the corresponding ENERGY-L algorithm at most $n$ times, finding at least one optimal $S_i$ value at every iteration. Hence, overall time complexity is $O(n^3)$.