The Power of Isolation

Silviu S. Craciunas Christoph M. Kirsch
Department of Computer Sciences
University of Salzburg, Austria
Email: firstname.lastname@cs.uni-salzburg.at

Abstract—**Non-functional system properties such as CPU and memory utilization as well as power consumption are usually non-compositional. However, such properties can be made compositional by isolating individual system components through over-provisioning. The challenge is to relate the involved isolation cost and the resulting isolation quality properly. Temporal and spatial isolation have been studied extensively. Here we study the compositionality of power consumption as another and in this regard unexplored example of a non-linear property with many important applications. In particular, we introduce the concept of power isolation for EDF-scheduled hard real-time systems running periodic software tasks. A task is power-isolated if there exist lower and upper bounds on its power consumption independent of any other concurrently running tasks. The main challenges in providing power isolation are to find a relationship between the power consumption of a system and the contribution of a single task to this power consumption as well as understanding the trade-off between quality and cost of power isolation. We present lower and upper bounds on the power consumption of a task as functions of task utilization, frequency scaling, and power model. Furthermore, we discuss the variance between lower and upper bounds (quality) and the power consumption overhead (cost) of power isolation.**

I. Introduction

Non-functional properties of software systems such as CPU and memory utilization as well as power consumption are non-compositional: they typically do not follow from individual components such as periodic software tasks due to the non-linear characteristics of hardware. Constructing large-scale software systems with respect to non-functional correctness is therefore difficult. However, non-functional properties can generally be made compositional by isolating individual components through over-provisioning. The challenge is to relate the involved isolation cost and the resulting isolation quality properly. Compositional scheduling [1] is a prominent example. In this paper, we study the compositionality of power consumption, which is another example of a non-functional property with significant importance in the embedded and non-embedded computing world.

Compositionality of system properties such as CPU and memory utilization is achieved through temporal and spatial isolation, respectively. Temporal isolation refers to techniques that prohibit tasks from altering the temporal behavior of other tasks. Spatial isolation refers to confining tasks to their own memory regions. We aim at studying compositionality of power consumption by isolating software tasks in terms of their individual power consumption. In particular, we provide lower and upper bounds on the power consumption of individual tasks independently of any other concurrently running tasks. The quality of power isolation is then given by the variance of per-task lower and upper power consumption bounds. The cost of power isolation appears as additional power consumed for providing different levels of power isolation quality.

Using earliest-deadline-first (EDF) [2] scheduling in power-aware systems we show that the power consumption of tasks can be bounded and thus isolated from other tasks without changing their real-time behavior. We also show that there is a trade-off between the quality and the cost of power isolation. Depending on the system and task properties, the penalty for improved quality of isolation, i.e., less variance between lower and upper power consumption bounds, is higher overall power consumption while tasks with certain properties can be isolated accurately with little power consumption overhead.

Many scheduling concepts that ensure temporal isolation, like CBS [3] and VBS [4] use EDF scheduling as the basic decision policy. The results presented in this paper may therefore be applied to such techniques, eventually enabling systems that provide full isolation in terms of time, space, and power. Furthermore, other non-functional properties which are non-compositional, such as thermal properties in large data centers [5], may also benefit from the analysis presented in the paper. Power isolation could also have applications in cloud computing [6]. The power consumed for processing the request of a client does not scale linearly with the speed of execution and, even worse, may depend on other, unrelated clients sharing the system. Time and power is therefore hard to account for in fair and competitive cost analyses. Power isolation may help to overcome fairness issues and, as a result, enable advanced pricing models for cloud computing.

We begin by presenting the task and power models that we study (Section II). We then discuss the quality of power isolation with different system settings (Section III) and address the cost of power isolation in Section IV. We present related work in Section V and conclude the paper in Section VI.

II. Task and Power Model

We consider the problem of power isolation in hard real-time systems using the periodic task model described by Liu and Layland [2]. Our task model description is similar to the one in [7]. Let $\Gamma = \{\tau_i \mid 1 \leq i \leq n\}$ be a set of n tasks

This work has been supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design and the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund S11404-N23).

with periodic activation. Each task τ_i is defined by a tuple (C_i, T_i, D_i), where C_i represents the computation time, T_i is the period, and D_i is the relative deadline of task τ_i. For simplicity we assume that $D_i = T_i$. Furthermore, each task τ_i produces an unbounded sequence of instances (also called jobs) $\tau_{i,k}, k = 1, 2, \dots$. As described in [7], each instance $\tau_{i,k}$ has a release time $r_{i,k} = (k-1)T_i$ and an absolute deadline $d_{i,k} = r_{i,k} + D_i$. The release time represents the time at which a job can be considered for scheduling and the absolute deadline represents the time at which the job must complete its workload of C_i time units. In this paper, we assume that there is no dependence between tasks, that the system allows preemption, and that the jobs $\tau_{i,1}$ are released at time zero ($r_{i,1} = 0$). The utilization U_i of a task τ_i is given by $U_i = C_i \backslash T_i$ and the total utilization of the tasks in the set Γ is $U = \sum_{i=1}^{n} U_i$. The jobs are scheduled using the preemptive earliest deadline first (EDF) [2] algorithm, a well-known optimal dynamic scheduling algorithm in which, at each time instant, the job that has the nearest absolute deadline is assigned to run on the CPU. In [2] the authors present a simple necessary and sufficient condition for the task set Γ to be schedulable under EDF, namely, if $U \leq 1$ then the set of n periodic tasks is schedulable by the EDF algorithm such that each job $\tau_{i,k}$ finishes its execution before its deadline $d_{i,k}$.

Reducing power consumption in real-time systems is widely studied [8], [9]. The main method in this context is dynamic voltage and frequency scaling (DVFS) [10]. In this paper we concentrate on the CPU energy consumption of a system which allows DVFS and do not analyze the energy consumption of other system components, like I/O devices. In [10] a simple DVFS mechanism for EDF-scheduled systems has been proposed. The mechanism builds upon the assumption that scaling the CPU operating frequency by a factor $\kappa \in (0, 1)$ will cause the execution time (but not the deadline and period) of a task to be scaled by a factor $1/\kappa$ [11], [10].

Note that this assumption depends on the particularities of the task. Whenever memory operations are involved, the execution time does not scale with the frequency [12], [13], and thus more accurate power-consumption models [8] or workload dissection [12] may be required.

The main result of [10] is that a set of periodic tasks remains schedulable if the system is scaled with a frequency scaling factor $\kappa = U$. Moreover, if the system is scaled to any frequency $f > \kappa \cdot f_{max}$, where f_{max} is the maximum available frequency, the system remains schedulable.

We take the power consumption characterization described in [14]. If the CPU is scaled to frequency f, the power consumption function of the CPU is

$$p(f) = c_0 + c_1 f^{\omega}, \tag{1}$$

where $\omega \geq 2$ is a constant, and the constants c_0 and c_1 represent the power consumption of the CPU in idle mode and at maximum frequency, respectively [14]. For all figures in this paper we use the power consumption function of an Intel XScale platform described in [15], [16], namely $p(f) = 1520 f^{\omega} + 80$ mWatt, with $\omega \geq 2$.

Since in an EDF-scheduled system we can scale the frequency at the beginning using the scaling factor $\kappa = U$ [10] (in the case of an ideal system with a continuous set of available frequencies), the energy consumption in an interval $[t_0, t_1)$ depends only on the sum of the utilizations of the tasks (U). The consumed CPU energy in an interval $[t_0, t_1)$ is

$$\int_{t_0}^{t_1} p(U \cdot f_{max}) dt.$$

We further discuss the quality and cost of power isolation in EDF-scheduled systems using the presented task and power model.

III. Quality of Power Isolation

In this section we start from the case when there are only two available frequencies, i.e., the processor is either in idle mode or running at the maximum frequency available. We then look at systems with a continuous set of available frequencies and at systems with multiple discrete frequency levels. In each case we provide upper and lower bounds for the CPU energy consumption of a task τ_i in the considered interval $[t_0, t_1)$ independently of the other tasks in the system. We choose the interval $[t_0, t_1)$ to represent the hyper-period of the periods of all tasks. Since we consider a time interval we will henceforth talk about CPU energy consumption. The average power consumption can be described by dividing the CPU energy consumption by the length of the considered time interval. We work with discrete time, i.e., the set of natural numbers $\mathbb{N}$ is the time-line.

We look at the CPU energy consumption of a system with total task utilization $U = \sum_{i=1}^{n} U_i$ running at frequency $\kappa \geq U$ in the interval $[t_0, t_1)$. In the considered interval the processor switches the frequency from κf_{max} when some task is running to 0 when no task is running. The CPU energy consumption over $[t_0, t_1)$ is

$$t_{idle} c_0 + t_{running}(c_0 + c_1(\kappa f_{max})^{\omega}),$$

where t_{idle} is the total time the processor is idle and $t_{running}$ is the total time some task is running, $t_{idle} + t_{running} = t_1 - t_0$.

Since $t_{idle} c_0 + t_{running} c_0 = (t_1 - t_0) c_0$, we can simplify the CPU energy consumption in the interval to

$$(t_1 - t_0) c_0 + t_{running} c_1 (\kappa f_{max})^{\omega}.$$

The base energy consumption E_{base} is the CPU energy consumption that is independent of the utilization of the tasks and is expressed as follows, $E_{base} = (t_1 - t_0) c_0$. Since the base CPU energy consumption does not depend on the utilization of the task set, we do not consider it in the discussion from this point on.

As elaborated in [10], if the frequency is scaled with the scaling factor κ, the computation time of all tasks will be scaled with factor $\frac{1}{\kappa}$. The total time some task runs in the considered interval can be written as

$$t_{running} = (t_1 - t_0) \sum_{i=1}^{n} \frac{C_i}{\kappa T_i} = (t_1 - t_0) \frac{U}{\kappa}.$$

We write the CPU energy consumption of the system in the considered time interval as

$$E(\kappa, U) = (t_1 - t_0)c_1 \frac{U}{\kappa}(\kappa f_{max})^{\omega}. \tag{2}$$

A. Two frequency levels

We first assume a system where there are only two possible discrete frequency levels, namely 0 and f_{max}. In this case we show that the CPU energy consumption of the periodic tasks in the system is compositional, namely, the CPU energy consumption of a single task τ_i with utilization $U_i = \frac{C_i}{T_i}$ is directly proportional to its utilization.

If there are only two frequency levels, the processor is scaled to the maximum frequency f_{max} when some task is running and to 0 when no task is running. Using Equation 2 we write the CPU energy consumption in the interval $[t_0, t_1)$ as

$$E(1, U) = (t_1 - t_0)c_1 U f_{max}^{\omega}.$$

It follows that the compositional CPU energy consumption function for a task τ_i with utilization U_i is

$$E(1, U_i) = (t_1 - t_0)c_1 U_i f_{max}^{\omega}.$$

Next we define the upper and lower bounds for the CPU energy consumption of a task τ_i as follows

$$bE_i^u = bE_i^l = E(1, U_i).$$

Note that in this case, the CPU energy consumption of a task is fully compositional in the sense that the amount of CPU energy that a specific task consumes is proportional to the amount of time it runs which in turn only depends on the utilization of the considered task. Thus we can write in this case that

$$E\left(1, \sum_{i=1}^{n} U_i\right) = \sum_{i=1}^{n} E(1, U_i).$$

With only two discrete frequency levels, the nonlinearity of the power consumption function is not expressed in terms of the utilization of the tasks and thus plays no role in the bounds.

An important aspect is the wake-up cost. In some systems the idle time must be sufficiently long such that the wake-up overhead is neutralized [17]. However, even in this case the amount of energy attributed to individual tasks does not change, just the overall CPU energy consumption.

Since there are only two frequency levels, the CPU energy consumption is higher than in the case of intermediate frequency levels. We compare the CPU energy consumed in this case to the ideal case where there is a continuous set of available frequencies in Section IV.

B. Continuous frequency levels

We now look at power isolation in an ideal system in which there is a continuous set of available frequency levels. In this case the frequency scaling factor, as described in [10], is $\kappa = U = \sum_{i=1}^{n} U_i$.

In order to isolate a task τ_i from the task set in terms of energy consumption we look at the two extreme cases, namely when the frequency is scaled from 0 to U_i and when the frequency is scaled from $1 - U_i$ to 1, where U_i is the utilization of task τ_i. Because of the nonlinearity of the CPU energy consumption function (Equation 1), a task τ_i has the lowest contribution to the CPU energy consumption if there are no other tasks running besides the given task and the highest contribution to the CPU energy consumption if by adding the task the utilization becomes 100%. The two cases are easy to prove using the binomial theorem. Given three utilizations $U_1, U_2, U_i \in (0, 1)$ with $U_1 \le U_2$, we have that

$$E(U_1 + U_i, U_1 + U_i) - E(U_1, U_1) \le \\ E(U_2 + U_i, U_2 + U_i) - E(U_2, U_2),$$

and conversely, for $U_1 \ge U_2$ we have

$$E(U_1 + U_i, U_1 + U_i) - E(U_1, U_1) \ge \\ E(U_2 + U_i, U_2 + U_i) - E(U_2, U_2).$$

The two cases occur when $U_1 = 0$ and when $U_1 + U_i = 1$, respectively.

Given the best and worst case we find the lower and upper bound on CPU energy consumption for a task τ_i with utilization U_i.

Equation (2) in this case gives the following CPU energy consumption function

$$E(U, U) = (t_1 - t_0)c_1(f_{max}U)^{\omega}.$$

We now look at the best case, namely that task τ_i with utilization U_i is the only task in the system. Hence the CPU energy consumption for this task, and therefore the lower bound on CPU energy consumption, is given by

$$bE_i^l = E(U_i, U_i) = (t_1 - t_0)c_1(f_{max}U_i)^{\omega}.$$

In the worst case, adding this task to a system will cause a frequency scaling from frequency $(1 - U_i)f_{max}$ to f_{max}. Therefore the upper bound on CPU energy consumption for task τ_i is

$$\begin{aligned} bE_i^u &= E(1,1) - E(1 - U_i, 1 - U_i) \\ &= (t_1 - t_0)c_1(f_{max}^{\omega} - ((1 - U_i)f_{max})^{\omega}) \\ &= (t_1 - t_0)c_1 f_{max}^{\omega}(1 - (1 - U_i)^{\omega}). \end{aligned}$$

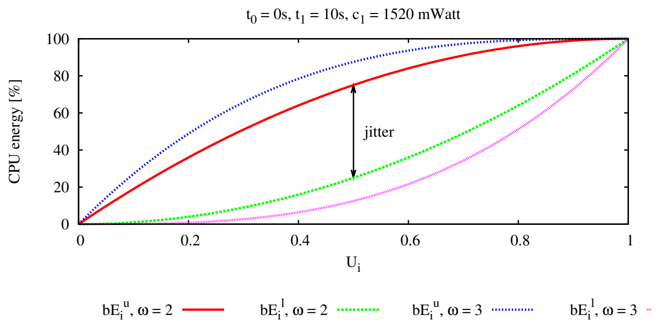


Fig. 1. CPU energy consumption bounds and jitter as a percentage of the maximum CPU energy consumption with $\omega = 2, 3$

Figure 1 shows the CPU energy consumption bounds with $\omega = 2$ and $\omega = 3$. The x-axis represents the utilization U_i of the task τ_i. The y-axis shows the CPU energy consumption as a percentage of the maximum CPU energy consumption, i.e., when running at frequency f_{max} for the whole considered time interval. Interestingly, the bounds converge for tasks with low and high utilization while the biggest difference between the upper and lower bound is when the task has utilization $U_i = 50\%$. Moreover, for $\omega = 2$ the difference between the bounds is less than for $\omega = 3$.

We measure the quality of power isolation through the CPU energy consumption jitter. The CPU energy consumption jitter jE_i is the difference between the upper and lower bound:

$$\begin{aligned} jE_i &= bE_i^u - bE_i^l \\ &= (t_1 - t_0) c_1 f_{max}^{\omega} (1 - (1 - U_i)^{\omega} - U_i^{\omega}). \end{aligned}$$

In practice a system does not have a continuous set of frequency levels. Hence, in the next section we discuss systems with discrete frequency levels.

C. Discrete frequency levels

In systems with discrete frequency levels, the frequency is scaled to the nearest frequency that is at least $f_{max} \cdot \sum_{i=1}^{n} U_i$. This ensures that all tasks will finish their computation within or at their deadlines.

Let $\kappa_1 \cdot f_{max} < \kappa_2 \cdot f_{max} < \cdots < \kappa_m \cdot f_{max}$ be the available frequency levels with $\kappa_1, \ldots, \kappa_m \in [0, 1]$, $\kappa_1 = 0$, $\kappa_m = 1$, and $m > 2$.

We first assume that we have three frequency levels, i.e., $m = 3$, namely 0, $\kappa \cdot f_{max}$, and f_{max}, where $\kappa \in (0, 1)$. Depending on the sum of the utilization of the tasks the frequency is scaled either to $\kappa \cdot f_{max}$ if $\sum_{i=1}^{n} U_i \leq \kappa$ or to f_{max} if $\sum_{i=1}^{n} U_i > \kappa$. Whenever the system is idle, i.e., no task is running, the processor is scaled to 0. Since we want to express the CPU energy consumption bounds of a task independently of any other task, we have no knowledge how the total system utilization will be affected by adding or removing task τ_i. Thus we have to consider the best- and worst-case contribution of task τ_i to the CPU energy consumption of the system. As seen in the previous subsection, where we considered continuous frequency levels, the worst-case contribution of a task τ_i to the CPU energy consumption is when by adding τ_i, the utilization becomes 100%. Conversely, the best case contribution of τ_i is when the task is the only one running in the system. This insight can be extended for the discrete frequency case. However, we need to prove whether the worst-case contribution of τ_i is when the frequency is switched from κf_{max} to f_{max}, or whether the worst-case contribution is when the frequency remains f_{max} with and without τ_i.

Proposition 1: The worst-case CPU energy consumption of a task τ_i is when, by adding the task to a system, the frequency is switched from κf_{max} to f_{max}, where $\kappa \in (0, 1)$.

Proof: We consider two possibilities. The first one is when the frequency is not scaled. Here, due to the nonlinearity of the power consumption function, the worst-case CPU energy consumption of a task τ_i is when by adding task τ_i, the utilization in the system increases from $1 - U_i$ to 1. The worst-case contribution to the CPU energy consumption of task τ_i in this case is $(t_1 - t_0) c_1 f_{max}^{\omega} (1 - (1 - U_i))$. The second possibility is when by adding task τ_i the frequency is switched from κf_{max} to f_{max} for some $\kappa \in (0, 1)$. We know that the utilization of all other tasks in the system satisfies

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} U_j \leq \kappa \text{ and} \sum_{\substack{j=1 \\ j \neq i}}^{n} U_j > \kappa - U_i.$$

The CPU utilization of all other tasks in the system besides τ_i is $\max(\kappa - U_i, 0)$. If the frequency is scaled from κf_{max} to f_{max} the utilization is either $\kappa - U_i$ in the case when $U_i < \kappa$ or 0 in the other case. The CPU energy consumption of the system without task τ_i at frequency level κf_{max} is

$$E(\kappa, \max(\kappa - U_i, 0)) > (t_1 - t_0) c_1 (\kappa f_{max})^{\omega} \max\left(1 - \frac{U_i}{\kappa}, 0\right).$$

With task τ_i the worst-case utilization of all tasks in the system is $\min(\kappa + U_i, 1)$, since the CPU energy consumption of the system cannot exceed the maximum possible CPU energy consumption, i.e., when the utilization is 100%, we take the minimum between $\kappa + U_i$ and 1.

Using Equation 2 the CPU energy consumption of the system is

$$E(1, \min(\kappa + U_i, 1)) \leq (t_1 - t_0) c_1 f_{max}^{\omega} \min(\kappa + U_i, 1).$$

We take the worst-case contribution of task τ_i to the CPU energy consumption when the frequency is scaled, which is always less than $E(1, \min(\kappa + U_i, 1)) - E(\kappa, \max(\kappa - U_i, 0))$.

We now have to prove that the CPU energy consumption in the first case is always lower than or equal to the CPU energy consumption in the second case, i.e.,

$$1 - (1 - U_i) \leq \min(\kappa + U_i, 1) - \max\left(1 - \frac{U_i}{\kappa}, 0\right) \kappa^{\omega}$$

There are four cases.

C.1 If $\min(\kappa + U_i, 1) = \kappa + U_i$ and $\max\left(1 - \frac{U_i}{\kappa}, 0\right) = 1 - \frac{U_i}{\kappa}$ then $U_i \leq \kappa + U_i - \left(1 - \frac{U_i}{\kappa}\right) \kappa^{\omega}$. After reductions, the inequality becomes $(\kappa - U_i)\kappa^{\omega - 2} \leq 1$, which is true since $\kappa \geq U_i$, $\omega \geq 2$, and $\kappa \in (0, 1)$.

C.2 If $\min(\kappa + U_i, 1) = 1$ and $\max\left(1 - \frac{U_i}{\kappa}, 0\right) = 1 - \frac{U_i}{\kappa}$ then $U_i \leq \frac{1 - \kappa^{\omega}}{1 - \kappa^{\omega - 1}}$, which is true since $\kappa^{\omega - 1} > \kappa^{\omega}$ for $\kappa \in (0, 1)$ and $\omega \geq 2$.

C.3 If $\min(\kappa + U_i, 1) = \kappa + U_i$ and $\max\left(1 - \frac{U_i}{\kappa}, 0\right) = 0$ then the inequality becomes $U_i \leq \kappa + U_i$, which is trivially true.

C.4 Last, if $\min(\kappa + U_i, 1) = 1$ and $\max((\kappa - U_i)\kappa^{\omega - 1}, 0) = 0$ then the inequality becomes $U_i \leq 1$, which completes the proof.

■

The best case contribution of task τ_i to the CPU energy consumption, i.e., the lowest CPU consumption of a system with task τ_i, is when τ_i is the only task in the system.

Given the worst- and best-case contribution of task τ_i to the CPU energy consumption we now give upper and lower bounds for the CPU energy consumption of task τ_i. We differentiate two cases depending on whether U_i is bigger or smaller than κ.

Case 1. **($U_i \leq \kappa$)** In this case, the lowest contribution to the CPU energy consumption of task τ_i is when by adding the task, the frequency level remains κf_{max}. Since the frequency is not scaled, the contribution is proportional to $U_i \backslash \kappa$.

We remind the reader that the term $U_i \backslash \kappa$ refers to how much CPU bandwidth the task τ_i will require after the system has been scaled with a scaling factor κ.

The lower CPU energy consumption bound is therefore

$$bE_i^l = (t_1 - t_0) c_1 \frac{U_i}{\kappa} (\kappa f_{max})^\omega = (t_1 - t_0) c_1 U_i f_{max}^\omega \kappa^{\omega - 1}.$$

The upper CPU energy consumption bound is

$$bE_i^u \quad = \quad (t_1 - t_0) c_1 f_{max}^\omega (\min(\kappa + U_i, 1) - \kappa^\omega (1 - \frac{U_i}{\kappa})).$$

We now elaborate on the upper CPU energy consumption bound. From Proposition 1, the worst-case contribution to the CPU energy consumption of task τ_i is when the frequency is scaled from κf_{max} to f_{max}. Naturally, for $U_i \leq \kappa$, the lowest possible utilization of all other tasks besides τ_i is $\kappa - U_i$. The CPU energy consumption without task τ_i at frequency level κf_{max} is

$$E(\kappa, \kappa - U_i) > (t_1 - t_0) c_1 (\kappa f_{max})^\omega \left(1 - \frac{U_i}{\kappa}\right).$$

The CPU energy consumption when running at frequency f_{max} is

$$E(1, \min(\kappa + U_i, 1)) \leq (t_1 - t_0) c_1 f_{max}^\omega \min(\kappa + U_i, 1).$$

The upper bound is thus the difference between the CPU energy consumption of all tasks including τ_i at level f_{max} and the CPU energy consumption at level κf_{max} of all tasks excluding τ_i, namely

$$\begin{aligned} E(1, \min(\kappa + U_i, 1)) \quad &- \quad E(\kappa, \kappa - U_i) \\ &< \quad (t_1 - t_0) c_1 f_{max}^\omega (\min(\kappa + U_i, 1) \\ &- \quad \kappa^\omega (1 - \frac{U_i}{\kappa})) \\ &= \quad bE_i^u. \end{aligned}$$

The CPU energy consumption jitter for $U_i \leq \kappa$ is

$$jE_i = bE_i^u - bE_i^l = (t_1 - t_0) c_1 f_{max}^\omega (\min(k + U_i, 1) - \kappa^\omega).$$

Case 2. **($U_i > \kappa$)** The difference between this case and the previous is that if there are no tasks in the system and task τ_i is added, the frequency is scaled to f_{max} directly. In this case only the lower bound changes to

$$bE_i^l \quad = \quad (t_1 - t_0) c_1 U_i f_{max}^\omega.$$

For $U_i > \kappa$, the utilization of all other tasks besides τ_i is 0 in the best case. The CPU energy consumption without task τ_i at frequency level κf_{max} is always greater than 0. The upper bound in this case is

$$bE_i^u = (t_1 - t_0) c_1 f_{max}^\omega (\min(\kappa + U_i, 1)).$$

We present several figures that show how the bounds and jitter behave dependent on the frequency level κf_{max}, the utilization of task τ_i, and ω.

In Figure 2 we plot the dependency of the CPU energy consumption jitter in terms of both κ and U_i. Figures 2(a) and 2(b) show a 2D map of the jitter as a percentage of the maximum CPU energy consumption with $\omega = 2$ and $\omega = 3$ respectively. The x-axis depicts U_i and the y-axis represents κ. We can see that for $\kappa < 0.2$ and $\kappa > 0.8$, the jitter remains at around $10 - 20\%$ independently of U_i. Similarly, for small and large task utilization, the jitter is around $10 - 30\%$ of the maximum CPU energy consumption. For $U_i \in [0.3, 0.7]$ and $\kappa \in [0.3, 0.7]$ we have the most jitter, i.e., the CPU energy consumption bounds have the largest variance.

We now look at the general case where there are multiple discrete frequency levels.

Let $\kappa_l \in \{\kappa_1, \kappa_2, \ldots, \kappa_m\}$ be the next higher frequency scaling factor to U_i such that $\kappa_{l-1} < U_i \leq \kappa_l$ and let $\kappa_u \in \{\kappa_1, \kappa_2, \ldots, \kappa_m\}$ be the next higher frequency scaling factor to $1 - U_i$ such that $\kappa_{u-1} < 1 - U_i \leq \kappa_u$.

The lowest contribution to the CPU energy consumption of task τ_i is when the task is the only task running in the system. Thus the processor is scaled to the frequency level given by $\kappa_l f_{max}$. The lower bound for task τ_i is therefore

$$bE_i^l = (t_1 - t_0) c_1 \frac{U_i}{\kappa_l} (\kappa_l f_{max})^\omega.$$

For the upper bound we consider the CPU energy consumption of task τ_i in the worst case in which adding the task to the system would scale the frequency from the largest frequency possible without τ_i in the system to f_{max}. Since we do not know the total utilization of the other tasks in the system, the frequency at which the system runs is the minimum between κ_u and κ_{m-1}. Let $\kappa = \min(\kappa_u, \kappa_{m-1})$.

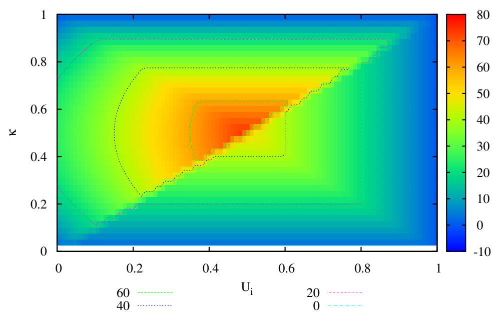
Following the same line of reasoning as in the case where there are only three available frequencies, the upper bound for task τ_i is

$$\begin{aligned} bE_i^u \quad &= \quad (t_1 - t_0) c_1 f_{max}^\omega (\min(\kappa + U_i, 1) \\ &\quad -\kappa^\omega \max(1 - \frac{U_i}{\kappa}, \min(\kappa_{u-1}, \kappa_{m-2}))), \end{aligned}$$

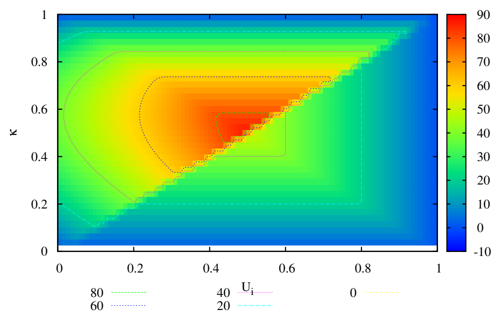
since

$$\begin{aligned} E(1, \min(\kappa + U_i, 1)) \quad &- \quad E(\kappa, \min(\kappa_{u-1}, \kappa_{m-2}))) \\ &< \quad (t_1 - t_0) c_1 f_{max}^\omega (\min(\kappa + U_i, 1) \\ &- \quad \kappa^\omega \max(1 - \frac{U_i}{\kappa}, \min(\kappa_{u-1}, \kappa_{m-2}))). \end{aligned}$$

The term $\min(\kappa_{u-1}, \kappa_{m-2})$ refers to the least amount of utilization that all other tasks besides τ_i in the system can have if the frequency is scaled from κf_{max} to f_{max} by adding τ_i.

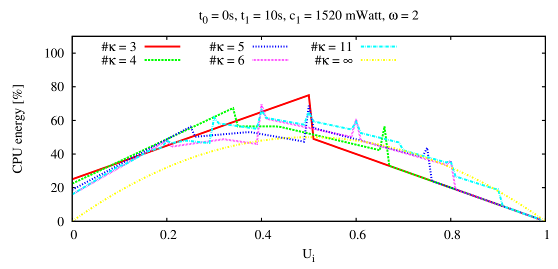


(a) $\omega = 2$

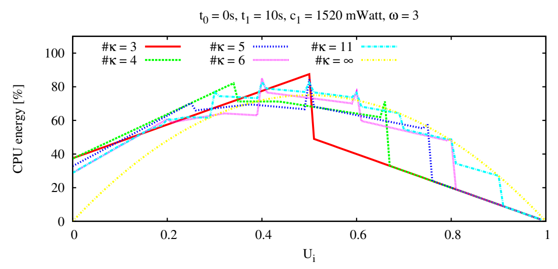


(b) $\omega = 3$

Fig. 2. CPU energy consumption jitter map as a percentage of the maximum CPU energy consumption in function of κ and U_i

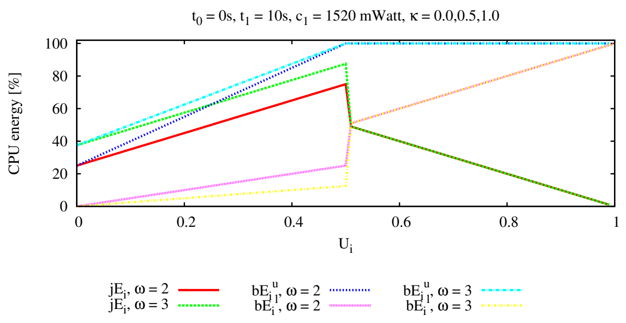


(a) $\omega = 2$

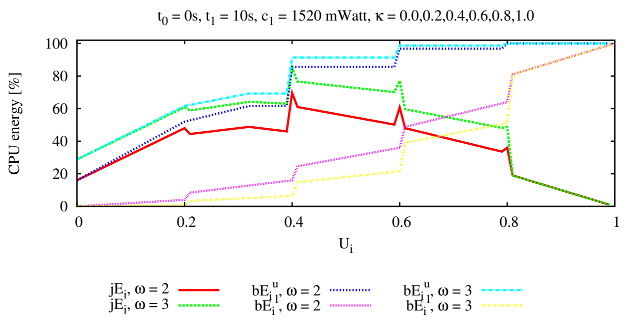


(b) $\omega = 3$

Fig. 3. CPU energy consumption jitter as a percentage of the maximum CPU energy consumption with different number of frequency levels



(a) 3 discrete frequency levels



(b) 6 discrete frequency levels

Fig. 4. CPU energy consumption bounds and jitter as a percentage of the maximum CPU energy consumption with multiple discrete frequency levels and $\omega = 2, 3$

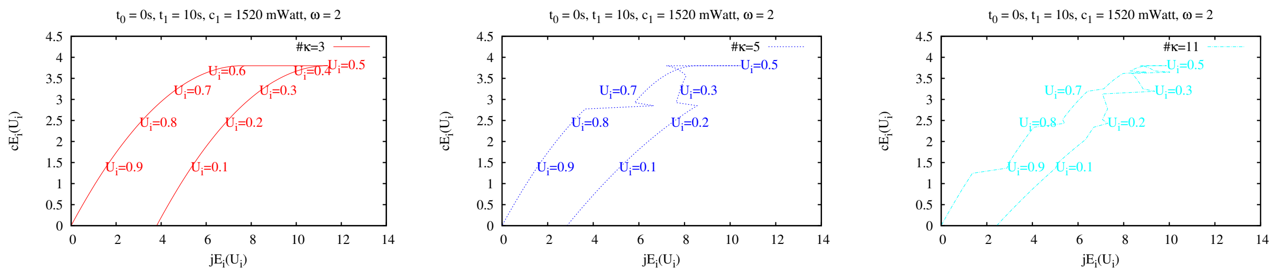
In Figure 3(a) and 3(b) we plot the CPU energy consumption jitter for all system configurations mentioned above and for $\omega = 2$ and $\omega = 3$, respectively. In addition we also plot an ideal system with a continuous set of available frequency levels. As before the x-axis is the task utilization and the y-axis represents the CPU energy consumption jitter as a percentage of the maximum CPU energy consumption.

In Figure 4 we plot the CPU energy consumption bounds and jitter as a percentage of the maximum CPU energy consumption (y-axis) in terms of the task utilization U_i (x-axis) for different system configurations and for $\omega = 2, 3$. In Figure 4(a) we have the frequency scaling factors $\{0, 0.5, 1\}$ and in Figure 4(b) we have the frequency scaling factors $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

IV. Cost of Power Isolation

We introduce the term of CPU energy consumption cost as the additional CPU energy consumed in the case of discrete frequencies over the ideal case where a system has continuous frequency levels.

As we have seen from the previous section, the quality of power isolation, i.e., the CPU energy consumption jitter,



(a) $\sharp\kappa = 3$ and $\omega = 2$ (b) $\sharp\kappa = 5$ and $\omega = 2$ (c) $\sharp\kappa = 11$ and $\omega = 2$

Fig. 5. Parametric plot of the CPU energy consumption cost and jitter as functions of U_i

is dependent on several factors. One factor is the number of frequency levels. We have seen that for two frequency levels the jitter is zero. With more frequency levels, the jitter increases but is also dependent on the utilization of the considered task τ_i. For tasks that have low or high utilization the jitter is less than for tasks with medium utilization. With a continuous set of available frequency levels, tasks with 50% utilization have the maximum jitter. We now quantify the cost of power isolation. Given the described power model, the lowest energy consumption is when the frequency is scaled to exactly the sum of the utilizations of the tasks, i.e., when there is a continuous set of available frequencies.

Note that, in some systems a race to idle, i.e., the CPU runs at maximum frequency in order to maximize idle time, is more energy efficient [18]. If this is the case the CPU energy consumption of the tasks is fully compositional as described in Section III-A. We analyze the general case where with increasing number of frequency levels the CPU energy consumption is closer to the ideal.

The cost of power isolation depends on four main factors, the sum of task utilizations, the utilization of the considered task, the number of available frequency levels, and the distribution of the frequency levels in the interval $[0, f_{max}]$.

We define the ideal CPU energy consumption in an interval $[t_0, t_1)$ for a set of tasks $\Gamma = \{\tau_i \mid 1 \le i \le n\}$ as the energy consumption in a system with a continuous set of frequency levels, namely $E_{ideal}(U, U) = (t_1 - t_0)c_1(f_{max}U)^\omega$.

The CPU energy consumption of a system with $\kappa_1 \cdot f_{max} < \kappa_2 \cdot f_{max} < \cdots < \kappa_m \cdot f_{max}$ frequency levels is

$$E_\kappa(\kappa_l, U) = (t_1 - t_0)c_1\frac{U}{\kappa_l}(\kappa_l f_{max})^\omega,$$

with $\kappa_l \in \{\kappa_1, \kappa_2, \ldots, \kappa_m\}$ such that $\kappa_{l-1} < \sum_{i=1}^{n} U_i \le \kappa_l$ being the frequency level closest to the total utilization of the tasks.

We define the cost metric as the additional CPU energy consumed in the case of discrete frequencies over the ideal case, namely $E_\kappa - E_{ideal}$.

We want to compare the quality and cost metric for a given task in isolation of any other tasks. Thus we have to bring the two metrics on a common denominator. Since the jitter depends on the utilization of the task τ_i, we also want to express the cost in terms of τ_i. If the total system utilization is 1 then the difference between E_κ and E_{ideal} is 0. The greatest difference in terms of cost is when only task τ_i runs in the system, but the system runs at maximum frequency. The best case CPU energy consumption is when only τ_i runs in the system and the frequency is scaled to $U_i f_{max}$. We can define an upper bound on the cost that is only dependent on τ_i, namely $cE_i = (t_1 - t_0)c_1 f_{max}^\omega U_i(1 - U_i^{\omega-1})$.

In Figure 5 we show the relation between the cost and quality of power isolation with different number of frequency levels ($\#\kappa$) uniformly distributed over $[0, 1]$[1]. Each figure is a parametric plot where the independent variable is the utilization of the task τ_i, the y-axis is the upper bound on the cost as a function of U_i, and the x-axis represents the CPU energy consumption jitter in function of U_i. While more frequency levels imply less isolation cost, the isolation quality is highly dependent on the task utilization and there is no strict monotonic relation between the number of frequency levels and the quality of isolation.

V. Related Work

We distinguish several directions of related research. The first is related to isolation of task behavior in real-time systems and more generally to power-aware real-time scheduling while still ensuring schedulability and temporal isolation. Another class of related research is centered on measuring and/or predicting the energy consumption of individual tasks in a system. Moreover, we also mention a third class of related results that enforce isolation of task power consumption.

In real-time systems the most important kind of isolation is temporal isolation. Temporal isolation is usually guaranteed via server mechanisms like CBS [3] and VBS [4] or through compositional scheduling [1]. Servers are usually defined by a period and a budget, where the budget is in most cases an upper limit on the time the server executes in the time-frame given by the period. Through dynamic voltage and frequency scaling (DVFS) the total CPU power consumption may be reduced without negatively affecting the timing restrictions of individual tasks [10], [20], [21]. Power-aware versions of server mechanisms have also been studied, e.g. [9]. The

[1]More figures depicting intermediary configurations can be found in [19].

aim of these approaches is to reduce the total CPU power consumption while maintaining temporal isolation. They do not, however, address isolating the tasks in terms of their individual power consumption.

Another line of research is concerned with measuring or approximating the energy consumption of specific tasks. In [22] the authors present a method for estimating the worst-case energy consumption (WCEC) of a task on a given platform. As the authors point out, the method is similar to WCET analysis because it computes upper bounds on energy consumption for each basic block in the control-flow graph of a task. In terms of measuring the power consumption of a task, a more recent example is [23] which proposes a system-call-based approach for fine-grained energy consumption accounting.

The work in [24] introduces a different approach to power isolation in the context of sensor networks. Each task is assigned a virtual energy source, called the virtual battery, with the aim to divide the total energy amongst all tasks (where each task can reserve a certain percentage). When a task has exhausted its allotted energy reserve it is terminated. Measuring and controlling power consumption has been studied for virtual machines [25], and in distributed systems (with a focus on thermal management) by employing the resource container abstraction [26]. In [27] the authors propose a model that, among other properties, allows power to be shared among tasks according to their proportional share. Another, related research direction is the Chameleon framework [28] in which power isolation is accomplished by letting each task specify and control their own power requirements while the system enforces isolation of these requirements. The presented methods from the third class of related research enforce power isolation, i.e., the behavior of tasks is changed in order to maintain power isolation. In contrast, our work provides a way to analyze the individual power consumption of tasks without intervening in the execution or scheduling of the tasks.

VI. Conclusion and Future Work

We have introduced the concept of power isolation for EDF-scheduled periodic hard real-time tasks and shown that such tasks can be effectively isolated with respect to their power consumption without affecting their relevant real-time behavior. In particular, we have provided lower and upper bounds on the individual power consumption of a given task independent of any other tasks in the system and shown that the quality of power isolation depends on task utilization and system properties related to frequency scaling. We have also discussed the cost of power isolation in terms of how much additional power may be consumed depending on system properties, power model, and quality of power isolation.

The analysis presented in this paper is a theoretical starting point for power isolation and compositionality. In future work we plan to address several assumptions that have been made, e.g., scaling cost, and use a more accurate power model [8].

References

[1] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. Embed. Comput. Syst.*, vol. 7, May 2008.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, 1973.

[3] L. Abeni and G. C. Buttazzo, "Resource reservation in dynamic real-time systems," *Real-Time Syst.*, vol. 27, no. 2, pp. 123–167, 2004.

[4] S. S. Craciunas, C. M. Kirsch, H. Payer, H. Röck, and A. Sokolova, "Temporal isolation in real-time systems: The VBS approach," *International Journal on Software Tools for Technology Transfer (STTT)*, 2012.

[5] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling "cool": temperature-aware workload placement in data centers," in *Proc. USENIX ATC*. USENIX Association, 2005.

[6] S. S. Craciunas, A. Haas, C. M. Kirsch, H. Payer, H. Röck, A. Rottmann, A. Sokolova, R. Trummer, J. Love, and R. Sengupta, "Information-acquisition-as-a-service for cyber-physical cloud computing," in *Proc. HotCloud*. USENIX, 2010.

[7] G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag, 2004.

[8] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memory energy usage under voltage scaling," in *Proc. EMSOFT*. ACM, 2007.

[9] C. Scordino and G. Lipari, "Using resource reservation techniques for power-aware scheduling," in *Proc. EMSOFT*. ACM, 2004.

[10] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. SOSP*. ACM, 2001.

[11] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proc. ICCAD*. IEEE Press, 2000.

[12] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proc. ISLPED*. ACM, 2004.

[13] V. Devadas and H. Aydin, "Real-time dynamic power management through device forbidden regions," in *Proc. RTAS*. IEEE, 2008.

[14] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *Proc. EMSOFT*. ACM, 2005.

[15] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems," in *Proc. ICCAD*. IEEE, 2007.

[16] J.-J. Chen and L. Thiele, "Expected system energy consumption minimization in leakage-aware DVS systems," in *Proc. ISLPED*. ACM, 2008.

[17] V. Devadas and H. Aydin, "On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications," in *Proc. EMSOFT*. ACM, 2008.

[18] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: understanding the runtime effects of frequency scaling," in *Proc. ICS*. ACM, 2002.

[19] S. S. Craciunas, C. M. Kirsch, and A. Sokolova, "The power of isolation," Department of Computer Sciences, University of Salzburg, Tech. Rep. 2011-02, July 2011.

[20] C. M. Krishna and Y. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," *IEEE Trans. Comput.*, vol. 52, no. 12, 2003.

[21] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *Proc. ASP-DAC*. IEEE Press, 2004.

[22] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the worst-case execution energy of embedded software," in *Proc. RTAS*, 2006.

[23] A. Pathak, Y. C. Hu, M. Zhang, V. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. Eurosys*. ACM, 2011.

[24] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzaher, "Virtual battery: An energy reserve abstraction for embedded sensor networks," in *Proc. RTSS*. IEEE Computer Society, 2008.

[25] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *Proc. USENIX ATC*. USENIX, 2007.

[26] A. Weissel and F. Bellosa, "Dynamic thermal management for distributed systems," in *Proc. TACS*, 2004.

[27] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Currentcy: a unifying abstraction for expressing energy management policies," in *Proc. USENIX ATC*. USENIX Association, 2003.

[28] X. Liu, P. Shenoy, and M. Corner, "Chameleon: application level power management with performance isolation," in *Proc. MULTIMEDIA*. ACM, 2005.