Technical Report

Optimal Procrastination Interval upon Uniprocessors

Muhammad Ali Awan
Patrick Meumeu Yomsi
Stefan M. Petters

CISTER-TR-130608
Version:
Date: 06-28-2013
Optimal Procrastination Interval upon Uniprocessors

Muhammad Ali Awan, Patrick Meumeu Yomsi, Stefan M. Petters

CISTER Research Unit
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340509, Fax: +351.22.8340509
E-mail:
http://www.cister.isep.ipp.pt

Abstract

Energy consumption is a major concern in modern real-time embedded systems and leakage current is a main contributor to it. To deal with the leakage current, several procrastination approaches have been proposed in the past in order to reduce the energy consumption. These approaches approximate the procrastination interval for the ease of analysis and sub-optimally utilise the potential to reduce the energy consumption. This paper presents an optimal method to determine the procrastination interval of each task and generalise the task-model that also covers the constrained deadline tasks. Analytical and experimental results show the superiority of the proposed techniques. In the best case, the proposed technique extends the average sleep interval up to 75% and decrease the energy consumption in idle state up to 55% over the state-of-the-art.
Optimal Procrastination Interval upon Uniprocessors

Muhammad Ali Awan  Patrick Meumeu Yomsi  Stefan M. Petters
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
{muaan,pamyo,smp}@isep.ipp.pt

ABSTRACT
Energy consumption is a major concern in modern real-time embedded systems and leakage current is a main contributor to it. To deal with the leakage current, several procrastination approaches have been proposed in the past in order to reduce the energy consumption. These approaches approximate the procrastination interval for the ease of analysis and sub-optimally utilise the potential to reduce the energy consumption. This paper presents an optimal method to determine the procrastination interval of each task and generalise the task-model to cover the constrained deadline tasks. Analytical and experimental results show the superiority of the proposed technique. In the best case, the proposed technique extends the average sleep interval up to 75% and decrease the energy consumption in idle state up to 55% over the state-of-the-art.

1. INTRODUCTION
Researchers have been studying uniprocessor embedded systems which consist of a finite number of recurring processes (referred to as “tasks” hereafter) for over forty years now. For such systems, each task is commonly characterized by parameters such as its worst-case execution requirement, its activation rate, and its temporal deadline reflecting its timing constraint. Over this period of time, they have come up with a number of very important results, develop some useful algorithmic techniques and built up an entire body of intuitions. Taken together, these results, techniques and intuitions have allowed system designers to come up with a very good understanding of the manner in which uniprocessor embedded systems behave. However, the emerging application requirements in the embedded systems arena have increased dramatically over the past years in terms of computing demands, need of reduced size and weight. Furthermore, besides having specific functional requirements, many embedded systems have stringent timing requirements (the system is then referred to as “real-time” (RT) embedded system). The RT embedded system domains include (but are not limited to) air-traffic control, aerospace, automotive, wind turbines, railway control systems, medical, factory automation, mobile phones and military equipment. Among these RT systems, hard RT systems are those for which violating any timing requirement can entail severe consequences, e.g., it can damage the system, lead to substantial economic loss, or even harm people or threaten human lives. Throughout this paper, hard RT embedded systems that have limited power supply are considered. This additional energy constrain is induced by battery power mobile device, limited or intermittent power supply for example. Even when the application is technically feasible upon the targeted platform in the sense that the platform can provide a sufficient computing capacity for the execution of the application, it has become unreasonable to expect to implement such a system without addressing the issue of minimizing its power and energy consumption. To this end, chip manufacturers are putting considerable efforts in this direction and this aim aligns neatly with the desired “wish-list” of most embedded systems.

There are two main sources of energy consumption in embedded systems: the dynamic power dissipation which is related to the current that flows when the switching of transistors takes place at runtime and the leakage power dissipation which is proportional to the current that flows regardless of gate switching. Since CMOS technology miniaturisation has increased the sub-threshold leakage current of modern processors exponentially to an extent where leakage power dissipation may dominate the dynamic power consumption, this factor can no longer be considered as negligible. This fact has been identified as a major concern in the International Technology RoadMap For Semiconductors 2010 Update under special topics [16]. To reduce the impact of leakage current, hardware vendors have provided multiple sleep states with reduced transition overheads (energy/time) when compared to previous processors, which can be exploited by the system designer to shut-down certain parts of the processor.

A well known approach used at system level to reduce the leakage power dissipation is called procrastination scheduling. The main idea behind this technique consists of delaying the execution of the processor already in sleep state as much as possible while ensuring the timing constraints of all tasks are met. As such, the number of sleep transitions are decreased and consequently the energy overhead is minimized. Many power saving algorithms based on procrastination scheduling [17,19,21] approximate the procrastination interval of tasks. This leads to sub-optimal energy savings. This research fills this gap.

The contribution of this paper is twofold. First, it presents an optimal method to compute the procrastination interval of the tasks for the implicit deadline task model and then it extends the results to a more general case, i.e., the sporadic constrained deadline task model where the temporal deadline of each task is allowed to be less than or equal to its activation rate. In sporadic task model, two consecutive instances of a task are separated by at least a minimum inter-arrival time.

One dimensional sensitivity analysis is used to show the optimality of the procrastination interval determined through the proposed method. It considers the so-called feasible region of the system in the C-Space and the fundamental notion of the allowance (“the maximum acceptable deviation of a task parameter” [20]) of the worst-case execution requirement of each task [11]. These two concepts are used together to compute the maximum allowance on top of the worst-case execution requirement of each task. The maximum allowances are used in turn to compute the maximum feasible delay that the system can undergo and finally, the delay...
is compared against the determined procrastination interval in the proposed method to show the optimality.

The rest of the paper is organized as follows. Section 2 and Section 3 present state-of-the-art and the system model used in this paper, respectively. Section 4 explains the limitations of the state-of-the-art while determining the procrastination interval and provides a new method to improve it over the existing solutions. The optimality of the proposed method along with its extension to the constrained deadline task model is also discussed in this section. The complexity of the proposed approach is presented in Section 5, which is followed by extensive simulation results presented in Section 6. The discussion is concluded in Section 7.

2. RELATED WORK

Leakage-aware scheduling was first addressed by Lee et al. [21] for periodic hard real-time systems. They proposed two different solutions: the leakage control earliest deadline first algorithm (LC-EDF) and the leakage control dual priority algorithm (LC-DP) for dynamic and static priority schemes, respectively. LC-EDF initiates the sleep state when the system becomes idle and delays the next busy interval to extend the sleep interval. This algorithm combines short idle intervals in the schedule to generate long sleep intervals and saves transition overheads. LC-DP works on the same mechanism for the static priority schedulers. The proposed algorithm needs external specialised hardware to manage such mechanism online. Baptiste [4] developed a polynomial time algorithm to minimise the static power consumption and transition overhead of the non DVFS system with unit sized RT apriori tasks.

Some efforts were made to combine the leakage-aware scheduling with DVFS to minimise the overall energy consumption. Irani et al. [15] considered shutdown in combination with DVFS and proposed a 3-competitive offline and a constant-complexity ratio online algorithm. They assume a continuous spectrum of available frequencies, an execution model with an inverse relation of frequency with execution time and an external hardware. Niu and Quan [25] addressed the dynamic and leakage power consumption simultaneously on a DVFS enabled processor for hard real-time systems. Their proposed algorithm is based on the latest arrival time of jobs estimated by expanding the schedule for the hyper-period. It cannot be used online due to extensive analysis overhead. The algorithm works with a given set of jobs with constrained deadlines. However, in real-time system it is common to have sequence of infinite job releases and their restrictive approach does not support such a model. Jejurikar et al. [19] improved LC-EDF and integrated it with DVFS to minimise the total power consumption. They also determined the critical speed \( q_{crit} \) that provides the lower bound on the processor frequency to minimize the energy consumption per cycle. They proved that the procrastination interval determined by their algorithm is always greater than or equal to the one estimated by the LC-EDF algorithm. Nevertheless, they did not relax on the requirement of the additional hardware.

Jeurjikar et al. [17] showed that LC-DP originally proposed by Lee et al. [21] may cause some of the tasks to miss their deadlines. They improved the original algorithm and combined it with their DVFS algorithm to reduce both dynamic and static power consumption. However, their system model is based on the same assumptions [19, 21]. Later on Chen and Kuo [7] determined some timing anomalies in the work of Jeurjikar et al. [17] and showed that their approach still might lead to some tasks missing their dead-

\(^1\)An algorithm is termed as competitive if its competitive ratio, i.e., ratio between the performance of the algorithm and the optimal offline algorithm, is bounded by a constant number.

lines. They proposed another two phase algorithm that estimates the frequency and the procrastination interval offline, and predicts shutdown instances online. The task not executing for their worst case execution time generates spare capacity in the schedule slack. The slack reclamation algorithm (SRA) of Jejurikar and Gupta [18] reclaims such slack which is used to further procrastinate or slow down the execution of the tasks to minimise the energy. Their proposed slack distribution policy either assigns entirely the dynamically reclaimed slack to slowdown or distributes it between slowdown and procrastination. SRA also needs an external hardware to implement the algorithm. Huang et al. [13, 14] estimated the procrastination interval for a device to activate the shutdown by predicting future events using RT calculus [27] and ensuring the system schedulability through RT interfaces [28].

The scope of this paper is the procrastination scheduling [18, 19, 21]. In this framework, the procrastination interval is revisited on the arrival of each task during the sleep interval. The estimated procrastination interval depends on the task-properties. This research aims to provide a new mechanism to reduce the pessimism involved in the state-of-the-art while estimating the procrastination interval and to extend the results to the sporadic task-model with constrained deadlines. Last but not least, the sensitivity analysis [2, 3, 29] provides tools to compute the tolerance over the execution time and the deadline reductions of the tasks. These techniques can also be tweaked and used to compute the procrastination interval of a task.

3. SYSTEM MODEL

The sporadic constrained deadline task model is assumed in this research, where a task-set \( \tau = \{\tau_1, \tau_2, \ldots, \tau_\ell\} \) is composed of \( \ell \) independent tasks. Each task \( \tau_i \) generates a potentially infinite sequence of jobs and is characterised by a 3-tuple \( \tau_i = (C_i, D_i, T_i) \), where \( C_i \) is the worst-case execution time, \( D_i \) is the relative deadline and \( T_i \geq D_i \) is the minimum inter-arrival time between two consecutive jobs of \( \tau_i \). These parameters are real-valued and given with the following interpretation. The \( k^{th} \) job \( j_{i,k} \) of \( \tau_i \) is defined as \( j_{i,k} = (r_{i,k}, c_{i,k}, d_{i,k}) \), where \( r_{i,k} \) is the absolute release time \((r_{i,k} - r_{i,k-1} \geq T_i)\), \( c_{i,k} \leq C_i \) is the absolute deadline and \( d_{i,k} = r_{i,k} + D_i \) is the absolute deadline. The hyper-period \( L^* \) of \( \tau \) is defined as the least common multiple of the tasks periods, i.e., \( L^* = LCM\{T_1, T_2, \ldots, T_\ell\} \). The notion of LCM is extended to real numbers as follows:

\[
LCM(a, b) \overset{\text{def}}{=} \inf\{x \in \mathbb{R}_+ : \exists p, q \in \mathbb{N}_+, x = pa = qb\}
\]

(see [6] for further details). The utilisation of task \( \tau_i \) is \( \mu_i \overset{\text{def}}{=} \frac{C_i}{T_i} \) and the system utilisation is \( \mu \overset{\text{def}}{=} \sum_{\forall\tau_i \in \tau} \frac{C_i}{T_i} \). Tasks are scheduled using the earliest deadline first algorithm (EDF) [22].

This work assumes a single processor which has active and idle states with power consumption of \( P_a \) and \( P_r \), respectively. A set of \( N \) sleep states with different characteristics is assumed. Each sleep state \( S_n \overset{\text{def}}{=} (P_n, tr_n, E_n) \), where \( P_n \) is the power consumption in the sleep state and \( E_n \) is the energy overhead associated to a complete sleep transition. A sleep state has the transition overhead time of going into a sleep state and the wake-up transition overhead from a sleep state to an active state. For the sake of simplicity, it is assumed these two transition overheads are equal and denoted as \( tr_n \). Each sleep state has a break-even-time \( bet_n \) computed through known approaches [1, 8, 9]. The definition of \( bet_n \) implies that the system will save energy if a sleep state \( S_n \) is initiated for more than
4. PROCRASTINATION INTERVAL

Initially, this work assumes implicit deadline task model i.e., $D_i = T_i, \forall \tau_i \in \tau$, to compare against the state-of-the-art which was designed for this model. Later in Section 4.5, this restriction is relaxed to a more general case, i.e., the constrained deadline task model, where tasks may have deadlines less than their periods ($D_i \leq T_i$). Formally, the procrastination interval is defined as follows.

**Definition 1 (Procrastination Interval).** The procrastination interval is the maximum time interval allowed to delay the execution of the ready tasks without violating any timing constraints of the system.

The longer duration of such an interval is desired in procrastination algorithms to reduce the energy consumption. Before presenting our procrastination technique, existing ones are discussed.

4.1 Limitations of the Existing Procrastination Approaches

In the leakage-aware procrastination scheduling, Lee et al. [21] initially proposed the online mechanism LC-EDF. To understand the basic principle behind this algorithm, let us consider the example given in Figure 1, taken from the work of Lee et al. [21]. Assume that task $\tau_k$ is the first which arrives in a sleep mode and has a deadline $D_k$. The procrastination interval $\Delta_k$ of $\tau_k$ is computed with the condition $\sum_{\tau_i \in \tau \setminus \{k\}} \frac{C_i}{T_i} + \frac{C_k + \Delta_k}{T_k} = 1$. Suppose $t$ is the current time then the timer is initialised with $t + \Delta_k$ to wake-up the system. After the timer initialisation, a procrastination interval is only recomputed when a newly arrived task has the highest priority compared to other tasks in the ready queue. For instance, after $\delta_k \leq \Delta_k$ time units, $\tau_i$ arrives with a deadline $D_k < D_i < D_k$; a new procrastination interval $t + \Delta_k$ is determined as $\sum_{\tau_i \in \tau \setminus \{k\}} \frac{C_i}{T_i} + \frac{C_k + \Delta_k}{T_k} + \frac{C_k + \Delta_k}{T_k} = 1$. The wake-up timer is reset to $t + \Delta_k$. Similarly, for any other task $\tau_j$ with the highest priority when compared to the tasks in the ready queue, the procrastination interval $\Delta_j$ in the sleep state of a processor is determined by using Equation 1, where $lp(j)$ is the indices of the tasks arrived before $\tau_j$ and have deadlines longer than $\tau_j$. In this equation, $\delta_j$ is the interval between an arrival of any job of task $\tau_j$ (having highest priority at that instant) and any next task arrival having priority higher than $\tau_j$ in the system’s sleep state. The limitations of LC-EDF are the increased online complexity to maintain a track of $\delta_j$ and considering the utilisation of the low priority tasks.

\[
\sum_{\tau_i \in \tau \setminus \{j\}, lp(j)} \frac{C_i}{T_i} + \sum_{i \in \text{lp}(j)} \frac{C_i + \delta_j}{T_i} + \frac{C_j + \Delta_j}{T_j} = 1 \quad (1)
\]

Jejurikar et al. [19] proposed an offline method to compute the procrastination interval for each task and thus reducing the online complexity. In the online phase, the first task that arrives in sleep mode initialises the wake-up timer $\zeta$ with its procrastination interval. The timer $\zeta$ counts down with every clock cycle. If another task (say $\tau_n$) arrives before the timer expires, the timer value is adjusted as follows: $\zeta \leftarrow \text{min}(\zeta, t + Z_k)$, where $t$ is the current time. They proposed Theorem 1 to estimate the procrastination intervals of the tasks offline, where $\eta_k$ is the frequency of the processor. The value of $\eta_k$ is set to 1, i.e., maximum frequency, for the ease of presentation. They also proved, it is superior to LC-EDF method to compute the procrastination intervals.

**Theorem 1.** [19] Given tasks in $\tau$ are ordered in non-decreasing order of their periods, the procrastination algorithm guarantees all task deadlines if the procrastination interval $Z_i$ of each task $\tau_i$ satisfies the following two conditions:

\[
\forall \tau_i \in \tau, \quad \frac{Z_i}{T_i} + \sum_{\forall \tau_k \in \tau \setminus \{i\}, k \leq i} \frac{C_k}{T_k} \leq 1 \quad (2)
\]

\[
\text{and} \quad \forall k < i, \quad Z_k \leq Z_i \quad (3)
\]

While computing the procrastination interval for task $\tau_j$, Jejurikar et al. [19] only considers the utilisation of the tasks having priority greater than or equal to $\tau_j$ (assuming a synchronous release of all tasks also known as critical instant in literature). Moreover, if any of the low priority task produce a low procrastination interval when compared to the high priority tasks, the procrastination interval of all the high priority tasks are readjusted by considering Equation 3. This latter equation is driven by the online approach of Jejurikar et al. (see [19] for details). Though the proposed method has its merits as it reduces the set of tasks considered for the procrastination of each task, its limitation is that it approximates the procrastination intervals by considering their utilisations. Let us demonstrate this shortcoming with the following example.

**Example 1:** Assume a task-set consisting of three tasks $\tau_1 = \langle 2, 4, 4 \rangle$, $\tau_2 = \langle 3, 7, 7 \rangle$ and $\tau_3 = \langle 0.25, 14, 14 \rangle$. Rearranging Equation 2, $Z$, can be computed with Equation 4 as given below.

\[
Z_1 = (1 - \frac{2}{2})4 = 2
\]

\[
Z_2 = (1 - \frac{2}{2} - \frac{3}{7})7 = 0.5
\]

\[
Z_3 = (1 - \frac{2}{2} - \frac{3}{7} - \frac{0.25}{14})14 = 0.75
\]

Final values after applying Equation 3 are $Z_1 = 0.5$, $Z_2 = 0.5$ and $Z_3 = 0.75$.

\[
Z_i = \left(1 - \sum_{\forall \tau_k \in \tau \setminus \{i\}, k \leq i} \frac{C_k}{T_k}\right) T_i \quad (4)
\]
4.2 Proposed Approach: Demand Bound Function Based Procrastination (PDBF)

The demand bound function (DBF) [5, 26] is used in this paper to compute the procrastination interval of the tasks in the context of uniprocessor scheduling. The DBF is an abstraction of the computation requirements of tasks which has been observed to correlate very closely with schedulability property of the task-set.

**Definition 2. (DBF [5]):** The demand for any constrained deadline task \( \tau_i \) and positive time \( t \), denoted by \( \text{DBF}(\tau_i, t) \), is the maximum cumulative execution requirement of jobs of task \( \tau_i \) in any interval of length \( t \). Formally, \( \text{DBF}(\tau_i, t) \) is presented in Equation 5.

\[
\forall t \geq 0, \quad \text{DBF}(\tau_i, t) \overset{\text{def}}{=} \left( \frac{t - D_i}{T_i} \right) + 1 \cdot C_i \tag{5}
\]

From Equation 5, it is easy to see that \( \text{DBF}(\tau_i, t) \) is a step-case function in \( t \) with first step occurring at time \( t = D_i \) and subsequent steps separated by exactly \( T_i \) time units. The DBF for the whole task-set is \( \sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t) \). The DBF based procrastination (PDBF) scheme achieves extended sleep intervals for any task-set. For instance, consider the DBF of the aforementioned example in Figure 3. Three stair case functions show the DBF of \( \tau_1 \), \( \tau_1 + \tau_2 \) and \( \tau_1 + \tau_2 + \tau_3 \). The straight line with a slope of 1 represents the supply bound function (SBF) of the processor.

PDBF uses the same logic as the one given in Theorem 1, i.e., synchronous release of all tasks sorted in a non-decreasing order of their deadlines and computes the procrastination interval of a task with DBF instead of considering tasks utilisation. Indeed when \( D_i \leq T_i \), the utilisation is not longer a good metric for the computation requirement of the tasks whereas the PDBF approach is easily extensible. To compute the maximum procrastination interval of a task \( \tau_i \), the PDBF approach subtracts the demand of the task \( \tau_i \) along with the demand of all the higher priority jobs from the SBF. It has to be noted that this difference is computed between the first deadline of task \( \tau_i \) and the end of the hyper-period (the reason is explained in Theorem 2). Due to the stair case property of the DBF, it is sufficient to compute the difference at the deadlines.

![Figure 3: Demand Bound Function of the example](image)

Theorem 2. Given tasks in \( \tau \) are ordered in a non-decreasing order of their relative deadlines, the PDBF scheme preserves all task deadlines, if the maximum procrastination interval of task \( \tau_i \), denoted by \( \chi_i \), is computed with Equation 7 while respecting the condition given in Equation 8.

\[
\chi_i \overset{\text{def}}{=} \min_{\forall \tau_j \in \tau : j \leq i, t \geq 0} \left( t - \sum_{\tau_k \in \tau : k \leq i} \text{DBF}(\tau_k, t) \right) \tag{7}
\]

where \( M = \left\{ n_j T_j : \frac{t}{T_j} \leq n_j \leq \left\lfloor \frac{t}{T_j} \right\rfloor \right\} \)

\[
\forall k < i, \chi_k \leq \chi_i \tag{8}
\]

**Proof Sketch.** Suppose a task \( \tau_i \) arrives in the sleep state. The timer is set to the procrastination interval computed with Equation 7 respecting the condition given in Equation 8. The time interval to wake up the system can only be decreased with an arrival of new task. This procrastination interval can be seen as additional task \( \tau_{\text{proc}} \) with a priority equal to the highest priority task, execution time equal to the wake-up sleep interval and it executes before the next busy period. Equation 8 ensures that all the tasks with deadlines greater than or equal to \( \tau_i \) will have procrastination interval greater than or equal to \( \chi_i \). Therefore, \( \tau_{\text{proc}} \) will not increase the system demand beyond the SBF in the presence of low priority tasks. Furthermore, the higher priority tasks can only shorten the execution time of \( \tau_{\text{proc}} \) (i.e., procrastination interval) on their arrival to respect their deadlines and the deadlines of the other tasks. The sleep interval is bounded by the procrastination interval of the first task and it only decreases with the new arrivals, therefore, based on the previous logic it will not affect any high priority task. Moreover, it is also sufficient to only consider the deadlines in an interval \([D_i, L_i] \) as the procrastination interval of a task is only considered when it has the highest priority on its arrival in the ready queue.

4.3 Procrastination Interval Improvement

The best known maximum procrastination interval is the one derived in Jejurikar et al. [19] method for each task in the state-of-the-art. This is objected by considering the worst-case scenario.
where DBF

**Case a)**

8 straight line is drawn between two points consideration. To compare these two approaches, their functions interval computed with . To prove the inequality given in Equation 9, we need to show that for all the deadlines between the first deadline of task and the hyper-period L∗, the procrastination interval computed with DBF is greater than or equal to Zt.Jejurikar et al. [19] computes Zt on the deadline of the task under consideration. To compare these two approaches, their functions are interpolated for all points in the demand bound function. To achieve this, let us consider the example depicted in Figure 2, a straight line is drawn between two points A(Tj, Ti ∑ Ci Tj) and B(L∗, L∗ ∑ Ci Tj) as shown in Figure 4. Note: Figure 4 only shows it for χ2 and Z2. The slope of this line is equal to ∑ Uk. In order to demonstrate that χi ≥ Zi, it is sufficient to prove this inequality in interval [Tj, L∗] (see Theorem 2). This interval is divided into two cases.

a) At time instances Ti and L∗, i.e., the deadline of τi and the hyper-period respectively.

b) An interval between time instance Ti and L∗, i.e., (A, B).

**Case a)** At the first time instant Ti, Equation 10 compares the two approaches.

\[ T_i - \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{T_i}{T_k} \right) C_k \geq \left( 1 - \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \frac{C_k}{T_k} \right) T_i \]

\[ \Leftrightarrow - \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{T_i}{T_k} \right) C_k \geq - \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \frac{C_k T_i}{T_k} \]

\[ \Leftrightarrow \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{T_i}{T_k} \right) \geq \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{C_k T_i}{T_k} \right) \]

\[ \Leftrightarrow \left( \frac{T_i}{T_k} \right) \leq \left( \frac{C_k T_i}{T_k} \right) \]

Equation 11 shows that at time instant T, Equation 9 holds. The same reasoning can be applied at time instant L∗ (i.e., by replacing the Ti with L∗ in Equation 10).

**Case b:** As already mentioned in the beginning of this proof, the demand of Jejurikar et al. [19] in an interval (A, B) is computed with a straight line of slope ∑ Uk and is compared against DBF at all deadlines. The equation of the line is  y = mx + c, where m is a slope and c is a y-intercept. The y-intercept is zero (i.e., c = 0) as the line passes through origin. Hence, the demand determined through the Jejurikar et al. [19] method is given in Equation 12.

\[ y = x \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \frac{C_k}{T_k} \]

Now consider any deadline that lies in between Ti and L∗ and then compare its y-coordinate to show that the demand of such deadlines lies below or on the line as the one given in Equation 12. Assume t ∈ M = {nji Tj : Tj < nji < L∗ Tj}. M describes the set of all the deadlines between Ti and L∗. As such nji Tj will be a deadline in an interval (A, B) and its demand is \[ \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \] (Equation 6). Let us put the deadline nji Tj in the x-coordinate of Equation 12 to get the resulting demand of Jejurikar et al. [19] and compare it against \[ \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \] as given in Equation 13.

\[ y = nji Tj \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \frac{C_k}{T_k} \geq \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \]

\[ \Leftrightarrow \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \geq \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \]

\[ \Leftrightarrow \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \geq \sum_{\forall k \in \tau_i : \tau_k \leq T_i} \left( \frac{nji Tj}{T_k} \right) C_k \]

Equation 14 is always true as x ≥ [x], ∀x. Thus, the curve of DBF is always below or on the line for all the deadlines in any interval (A, B).

As the demand of Jejurikar et al. [19] method for all deadlines in the interval [A, B] (case a and b) are greater than or equal to DBF, the lemma follows.
4.4 Minimum Idle interval Improvements

The minimum bound on the idle period in the schedule is an important metric in procrastination scheduling to select the most efficient sleep state, $S_n$, offline. It is the length of the shortest idle interval in the schedule, and all the idle intervals will be greater than or equal to this bound. To reduce the online complexity, a processor can choose its sleep state based on this interval that minimises the energy consumption in sleep state while respecting the temporal constraint. By maximising the minimum bound on the idle period, system increase the chance to use the better sleep states (when system has more than one sleep state [1]) which in turn reduces the energy consumption. Therefore, it is also important to maximise the minimum bound on the idle interval.

**Lemma 2.** Given tasks in $\tau$ are ordered in a non-decreasing order of their relative deadlines, the minimum idle period guaranteed by PDBF scheme is always greater than or equal to the minimum idle period guaranteed by Jejurikar et al. [19].

**Proof.** Assume all the tasks are sorted in non-decreasing order of their periods/deadlines. The minimum procrastination interval $Z_{\min}$ determined through Jejurikar et al. [19] algorithm is equal to $Z_{\min} = \min_{\forall r_{j} \in \tau} \chi_j$. Similarly, the minimum idle period guaranteed by the PDBF scheme $\chi_{\min} = \min_{\forall r_{j} \in \tau} \chi_j$. To prove the above mentioned lemma, one needs to prove Equation 15.

$$\min_{\forall r_{j} \in \tau \in M} \left( 1 - \sum_{\forall r_{j} \in \tau \subset \lambda} \frac{1}{T_j} \right) C_i \geq \min_{\forall r_{j} \in \tau \subset \lambda} \left( 1 - \sum_{\forall r_{j} \in \tau \subset \lambda} \frac{C_j}{T_j} \right) T_i,$$

where $M = \left\{ n_j T_j : \frac{\min_{\forall r_{j} \in \tau \subset \lambda} T_j}{T_j} \leq n_j \leq \frac{L^*}{T_j} \right\}$

In order to prove this inequality, we have to show that for $t \leq L^*$ the demand of the given task-set will remain below or will be equal to the demand computed by Jejurikar et al. [19] method, where $t \in M = \left\{ n_j T_j : 1 \leq n_j \leq \frac{L^*}{T_j} \right\}$. In other words, all the deadlines are checked for the difference. To interpolate the demand computed by Jejurikar et al. the demand on the neighbouring deadlines of a task are connected with a straight line. Finally, the demand beyond the last period is extended with a line having a slope equal to the system utilisation. For instance, Figure 5 shows the demand of the given example with both DBF and the procrastination algorithm proposed by [19]. For Jejurikar et al. [19] algorithm, the demand of the tasks computed on their first deadline are represented with $A$, $B$ and $C$ points. Points $A$ and $B$ are connected with a straight line to compare against all the deadlines in the DBF happening in between these two points. Similarly, $B$ and $C$ points are connected, and the demand beyond $C$ for procrastination algorithm is extended with a line having a slope equal to the utilisation of the task-set.

Since the DBF needs to be checked at more instances than $A$, $B$ and $C$ in the procrastination algorithm, we need to consider constraints. The objective is to find the minimal distance with the unity line and the demand. For all intervals between successive points $A$, $B$ and $C$, it is true that the smallest gap between the unity line and the demand within this interval of those can be found in either of the two delimiting points (for example, for interval $[A, B]$, the smallest gap can either occur at $A$ or $B$). Since $U \leq 1$, and it is evident that beyond the largest period, the largest gap can be found at the largest period point. In order to demonstrate that the gap computed with the DBF based value is always greater than or equal to that of procrastination algorithm [19] it is sufficient to show that the DBF test dominates in the following cases.

**Case a** To get the first deadline of every task, we set $t = T_i$ in Equation 15.

**Equation 16** shows for the first case that Equation 15 holds.

**Case b** The demand computed by the DBF is always smaller than the connecting lines of the first deadline of all tasks.

\[ \min_{\forall r_{j} \in \tau \subset \lambda} \left( 1 - \frac{T_i}{T_j} \right) C_i \geq \min_{\forall r_{j} \in \tau \subset \lambda} \left( 1 - \frac{C_j}{T_j} \right) T_i \]

\[ = \frac{T_i}{T_j} \sum_{\forall r_{j} \in \tau \subset \lambda} \frac{C_j}{T_j} \leq \frac{T_i}{T_j} \sum_{\forall r_{j} \in \tau \subset \lambda} \frac{T_j}{T_j} C_j \]

\[ = 0 + \sum_{\forall r_{j} \in \tau \subset \lambda} \frac{T_j}{T_i} C_j \]

\[ \leq \frac{T_i}{T_j} \]

\[ \text{Equation 16 shows for the first case that Equation 15 holds.} \]
Equation 17 is the general form of the equation of a line between two points \((x_1, y_1)\) and \((x_2, y_2)\). In the representation of the DBF, the \(x\)-axis and \(y\)-axis represent the time and the demand, respectively. Let us assume the coordinates at the deadlines of \(\tau_{i-1}\) and \(\tau_i\) are \((x_1, y_1) = \left(T_{i-1}, \sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_{i-1}\right)\) and \((x_2, y_2) = \left(T_i, \sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_i\right)\), respectively. To find the equation between these two points, substitute their coordinates into Equation 17 correspondingly as shown in Equation 18.

\[
y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1
\]

(17)

\[
y = \left(\sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_{i-1}\right)(x - T_{i-1}) + \sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_{i-1}
\]

(18)

Now consider any deadline that lies in between the deadlines of \(\tau_{i-1}\) and \(\tau_i\) (i.e., between \((x_1, y_1)\) and \((x_2, y_2)\)). It is shown that the demand (y-coordinate) of such deadlines will be below or on the line given in Equation 19. To this end, let us say that any deadline between \((x_1, y_1)\) and \((x_2, y_2)\) is specified by \((x_m, y_m)\) in Equation 19 and compare the resulting value of the y-coordinate with its \(y_m\). If it is greater than or equal to \(y_m\), then DBF is below or on the line. The resulting expression is shown in Equation 20.

\[
n_iT_j\left(\sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_{i-1}\right) - \frac{C_k}{T_k} (x_m - T_{i-1}) \geq \sum_{\forall y_k \in \mathbb{S} \leq i} \frac{n_iT_j}{T_k} C_k
\]

(19)

Hence, it follows that

\[
n_iT_j\left(\sum_{\forall y_k \in \mathbb{S} \leq i} \frac{C_k}{T_k} \tau_{i-1}\right) - \frac{C_k}{T_k} (T_{i-1} - T_{j-1}) \geq \sum_{\forall y_k \in \mathbb{S} \leq i} \frac{n_iT_j}{T_k} C_k
\]

(20)

Point \((x_m, y_m)\) is in between \(T_{i-1}\) and \(T_i\), therefore the factor \(n_iT_j C_k\) can be rewritten as \(\sum_{\forall y_k \in \mathbb{S} \leq i} \frac{n_iT_j}{T_k} C_k\).

4.5 Extensions to the constrained deadline task model

The state-of-the-art procrastination algorithms [19,21] cannot be extended for constrained deadline task model \((D_i \leq T_i)\) in their current form. One of the advantages of the PDBF approach is that it straight forward extension to this model. For the constrained deadline task model, Equation 7 can be rewritten in its general form by replacing \(DBF(t_i, t_j)\) with \(DBF(t_i, t_j)\) as given Equation 22, where the set \(M\) is substituted by

\[
M_1 = \left\{ n_iD_j : \left| \frac{T_i - D_i}{D_i} \right| + 1 \leq n_j \leq \left| \frac{L^* - D_i}{D_i} \right| + 1 \right\}
\]

Similarly, the minimum idle interval for constrained deadline task model is given in Equation 23, where

\[
M_2 = \left\{ n_iD_j : 1 \leq n_j \leq \left| \frac{L^*}{T_j} \right| \right\}
\]

\[
\alpha = \frac{1}{\max\left(U, \sup_{t \in [D_{\text{min}}, L^*]} \frac{DBF(t_1)}{t}\right)}
\]

(24)

**Corollary 1**. \(\alpha \geq 1\) for a feasible system.

**Proof**. Assuming a feasible system by EDF upon a uniprocessor platform, \(U \leq 1\). Moreover, \(DBF(t_1, t) \leq t \frac{DBF(t_1)}{t} \leq 1\). Therefore, \(\frac{DBF(t_1)}{t} \leq \frac{DBF(t_1)}{t} \leq 1\). As both quantities in the denominator of \(\alpha\) are less than 1, it follows that \(\alpha \geq 1\).
Algorithm 1 Minimum Idle Interval Determination

Inflation Phase:
1. Compute $\alpha$ with Lemma 3.
2. Inflate each job by $(\alpha - 1)C_i$.
3. Generate a schedule for its hyper-period

Deflation phase:
4. for all Jobs from last to first (with respect to execution) do
5. Suppose the selected job is $j_{i, n}$ and $t_s$ is the start time of its execution.
6. for all Jobs executing before $t_s$ do
7. Suppose $j_{i, n}$ is the job executing before $t_s$
8. Move $j_{i, n}$, by $\min\{(\alpha - 1)C_i, \gamma\}$, where $\gamma$ is the difference between $d_{i, n}$ and time instant where $j_{i, n}$ starts its execution.
9. end for
10. end for

Packing Phase:
11. for all Jobs from First to Last do
12. if First Job then
13. Move job to the left as much as possible respecting its release instant
14. end if
15. end for
16. Assume, synchronous release happens at time instant $t_{sync}$ and first instant of execution starts at time instant $t_{first}$, then $\chi_{min} = t_{first} - t_{sync}$

an extra allowance of $\alpha C_i - C_i = (\alpha - 1)C_i$ to the WCET of each job. This work proposes an algorithm that exploits the scaling factor $\alpha$ to determine the minimum idle interval and then later on shows its optimality. The pseudo-code is given in Algorithm 1. It can be divided into three phases. 1) Inflation Phase: All the jobs in the hyper-period are inflated in this phase with extra allowance to achieve the maximum utilisation while keeping the system schedulable. 2) Deflation Phase: In this phase, each job deflates and allows the schedule on its left to shift right by a maximum time interval equal to the deflation of the previously visited jobs. All the deadlines are respected and jobs are tackled one by one starting from the last job towards the first one in the complete hyper-period. The maximum difference between the first job from the first execution instant and time of synchronous release gives the minimum idle interval. 3) Packing Phase: This phase is applied on the shifted schedule developed in the deflated phase, as a work-conserving scheduler (EDF) is used in this work. All the other jobs excluding the first one are shifted from the right to the left respecting their release times.

To elaborate on the given algorithm let us consider an example of a system consisting of three tasks $\tau_1 = \{1, 4, 5\}, \tau_2 = \{1, 4, 6\}$ and $\tau_3 = \{1, 7, 10\}$. The task-set has $\alpha = 2$. The different phases of the algorithm are illustrated in Figure 6. In the inflation phase, all the tasks inflate their execution by $(\alpha - 1)C_i$. It is evident in the inflation phase that an addition of any $\epsilon > 0$ to the $C_i$ of any task will make system unschedulable. In the deflation phase, all the jobs are deflated and moved to the right respecting their deadline constraint. The packing phase considers the work-conserving property of the scheduler and moves all the jobs towards left while adding a delay of $\chi_{min}$ after the synchronous release.

**Theorem 3.** The minimum idle period determined with Algorithm 1 for a constrained deadline task-set is optimal.

The processor is never idle when there is a job in the ready queue.

**Proof.** $\alpha$ gives the maximum allowance to a job to extend it execution, which is optimal by Lemma 3. In the inflation phase, any job cannot be further inflated by any $\epsilon > 0$ without missing a deadline. In the deflation phase, Algorithm 1 shifts the schedule to the right respecting all the deadlines by choosing $\min\{(\alpha - 1)C_i, \gamma\}$ that guarantees that none of the deadlines are missed during this process. As the schedule cannot be shifted for more than the sum of inflated time interval in the system, therefore, by construction, this leads to $\chi_{min}$, which is then the optimal minimum idle interval in the whole schedule.

The procrastination interval of each specific task $\tau_i$ can be determined with the above mentioned Algorithm 1 by considering the task along with those with a higher priority. The determined procrastination interval should satisfy the condition given in Equation 8. Furthermore, the value of $\alpha$ mentioned in Equation 24 is determined for the whole task-set. In this case, tasks having a priority higher than or equal to the task under consideration are considered. The modified form of $\alpha$ mentioned in Equation 25 should be used while determining the individual procrastination interval of the task with Algorithm 1,

$$\alpha \overset{def}{=} \max\left\{ \sum_{\tau_k \in \tau \setminus \{\tau_i\}} U_j, \sup_{t \in [D_{min}, L^*]} \frac{\sum_{\tau_k \in \tau \setminus \{\tau_i\}}}{\tau_k} \right\}$$

**Theorem 4.** Let $\tau$ be a constrained deadline task-set. The procrastination interval for any task in $\tau$ determined through Algorithm 1 is optimal.

**Proof.** The proof directly follows from Theorem 3.

**Corollary 2.** The procrastination interval computed with Algorithm 1 is equivalent to PDBF.

**Proof.** The minimum idle interval $\chi_{min}$ computed with Algorithm 1 is the maximum procrastination interval that a system can achieve after the synchronous task release. Any additional time $\epsilon > 0$ to this interval will cause a deadline miss. Similarly, the
minimum idle interval computed with PDBF also does not further allow to procrastinate after the synchronous release. Hence, these two intervals are equivalent, which in turn also shows the optimality of PDBF.

The same logic mentioned in above corollary holds for the procrastination interval of the individual tasks.

5. COMPLEXITY COMPARISON

The complexity of the state-of-the-art approaches as well as that of the proposed approach to compute the procrastination interval can be categorised as offline and online complexity. The offline complexity of the LC-EDF algorithm [21] is zero as all the computations are performed online. Jejurikar et al. [19] methods has an offline complexity of $O(\ell^2)$. The PDBF approach has an offline complexity of $O(\ell h)$, where $\ell$ is the number of jobs in the hyper-period.

The online complexity of the PDBF approach and Jejurikar’s method is the same and equals to $O(\ell)$. The LC-EDF algorithm has an online complexity of $O(\ell \alpha)$. This implies that the external hardware designed for Jejurikar’s method can also be used for the PDBF approach as both work on the same principle. On the one hand, the procrastination based energy saving algorithms proposed for Jejurikar’s method can be easily integrated with the PDBF approach without any extra effort. On the other hand, the LC-EDF algorithm needs complex external hardware due to the mechanism used to compute procrastination interval online.

6. EVALUATION

The discrete event simulator SPARTS (Simulator for Power Aware and Real-Time Systems) [23, 24] is used to evaluate the effectiveness of the PDBF approach. SPARTS is used with the parameters mentioned in Table 1, where underlines values are the default values if not mentioned in the description of the experiment. The parameters $C_h^g$ and $\Gamma$ are used to generate wide variety of different tasks and their subsequent varying jobs. Suppose, $C_h^g$ and $\Gamma$ are the helper variables to provide the bounds on the best-case execution time (BCET) and sporadic delay of a task respectively. Then $C_h^g$ and $\Gamma$ are randomly selected for the given tasks in interval $[C_h^g - C_h^s, C_h^g]$ and $T_i$, $[\Gamma, 1]$ respectively. Similarly, the actual execution time and sporadic delay of the individual jobs are randomly selected from the following intervals $[C_h^g, C_i^g]$ and $[T_i, T_i + \Gamma]$ respectively. The periods of the task-set are chosen from an interval, $T_{\min}$, PUB, where $T_{\min}$ it the lower bound and PUB (Period Upper Bound) is the variable used to define the upper bound on the interval. Each task-set with different parameters mentioned in Table 1 is simulated for 100 times with different seed values to the random number generator and averaged. The simulation time of each task-set is 100sec.

The SRA algorithm [18] is an energy saving approach that takes procrastination intervals of the tasks determined through Jejurikar’s method as an input. For a fair comparison, the same algorithm is used by just replacing the input phase with PDBF determined procrastination intervals. For simplicity sake, it is assumed that all the slack in the schedule (spare capacity) is reserved for the shutdown of the processor. Both variation of SRA is implemented in SPARTS and their sleep state is selected offline based on their respective minimum idle interval. It has already been shown in the state-of-the-art that SRA performs better than LC-EDF, hence, this section restrict the comparison to SRA.

The power model used for simulations is based on the Freescale PowerQUICC III Integrated Communications Processor MPC8536 [10]. The power consumption values are taken from its data sheet for different modes (Maximum, Typical, Doze, Nap, Sleep, Deep...).
its frequent switching. In the best case, the average increase in the

It has been shown in their data sheet, therefore, the transition overhead are

assumed for four different sleep states. The transition overhead of the
typical mode that corresponds to the idle state in our system model is
considered negligible. The power values given in Table 2 sum up core power and platform power consumption. More details are available
in the reference manual [10].

Figure 7 presents the gain of PDBF over SRA with respect to average sleep interval for different values of $U$ and PUB. The average sleep interval is computed by accumulating the idle time in the scheduling and dividing it by the number of sleep states. The gain of PDBF increases with an increase in system utilisation. Furthermore, the gain also increases by widening the interval to select $\tau_s$ of the tasks. At low utilisation PDBF and SRA have enough slack to initiate longer sleep intervals. However, with an increase in system utilisation, the slack in the system decreases, and the procrastination intervals lengths have a high impact on the sleep intervals. Another reason for a high gain at high utilisation is the difference of minimum idle interval between SRA and PDBF. It has been shown in Lemma 2 that $\chi_{min} \geq Z_{min}$. Therefore, SRA starts to lose efficient sleep states at high utilisation, causing its frequent switching. In the best case, the average increase in the

sleep interval is approximately 75%.

The gain in average sleep interval is also computed by varying the utilisation against the BCET Limit $C^b$ as shown in Figure 8. Mostly, the gain occurs due to an increase in system utilisation, while the variation in $C^b$ has a minute effect at a very high utilisation of 0.95. As both algorithms use the same mechanism to manage the slack, the difference is negligible. The change in sporadic delay limit $\Gamma$ is also observed in the experiments against different values of $U$. The effect of $\Gamma$ is negligible as well. The variation in task-set size is demonstrated in Figure 9 against different values of $U$. In the best case (i.e., $|\tau| = 100$), the gain reaches to 75%. It is evident that the increase in task-set size increases the gain in average sleep interval. This can be explained as follows. The procrastination interval of a high priority task is always bounded by its

delay limit $\Gamma$.
The difference comes in the energy consumption of the system in idle intervals and termed as reducible energy consumption (REC). The gain of PDBF over SRA with respect to REC is compared for different parameters against system utilisation as demonstrated in Figure 10, Figure 11 and Figure 12. In the best case, the gain in REC is approximately 55%. All the graphs have more or less similar trends as explained in the description of their corresponding results with average sleep intervals.

7. CONCLUSIONS

The PDBF technique is optimal to compute the procrastination intervals of a given task-set. It has been shown theoretically and experimental that PDBF dominates over SRA. The average sleep interval can be increased up to 75%, while the REC can be raised up to 55%. The online complexity of PDBF is the same when compare to that of SRA. The relaxation to the constrained deadline task model is an additional benefit of the proposed approach. In the future, it is intended to extend this procrastination algorithm to heterogeneous multicore platforms and also for the dependent task-set model.

Acknowledgements

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme ‘Thematic Factors of Competitiveness’), within [REPO-MUC] project, ref. FCOMP-01-0124-FEDER-015050, and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/70701/2010.

8. REFERENCES


