Leveraging Performance and Power Savings for Embedded Systems using Multiple Target Deadlines*

Edward Y. Y. Kan
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
edkan@cs.hku.hk

W. K. Chan†
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

T. H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract* — Tasks running on embedded systems are often associated with deadlines. While it is important to complete tasks before their associated deadlines, performance and energy consumption also play important roles in many usages of embedded systems. To address these issues, we explore the use of Dynamic Voltage and Frequency Scaling (DVFS), a standard feature available on many modern processors for embedded systems. Previous studies often focus on frequency assignment for energy savings and meeting definite task deadlines. In this paper, we present a heuristic algorithm based on convex optimization techniques to compute energy-efficient processor frequencies for soft real-time tasks. Our novel approach provides performance improvements by allowing definitions of multiple target deadlines for each task. We simulate two versions of our algorithm in MATLAB and evaluate their performance and efficiency. The experimental results show that our strategy leverages performance and energy savings, and can be customized to suit practical applications.**

Keywords — DVFS; multiple deadlines; power savings; energy; convex optimization

I. INTRODUCTION *†

The development of embedded systems with real-time constraints should consider not only the traditional aspects such as effective task scheduling and accurate execution time prediction, but also the environmental horizon such as minimizing energy consumption while providing sufficient performance to users. Consider a user operating a wireless barcode scanner running on a real-time OS. Such a scenario frequently occurs during stocktaking in a supermarket. Provided with a legible piece of barcode and an operational profile of barcode scanning, a deadline may be defined for the task of reading the barcode. If the deadline is missed, the scanner is deemed to be unable to recognize the barcode. Though undesirable, occasional deadline violations may be tolerated; hence, this kind of deadline is called a soft deadline. If we attempt to increase the power of the laser beam module and the decoding processor to shorten the scanning time, more cycles can be completed, which may lead to an improvement in operational efficiency. On the other hand, if the scanning time is too short, the collaborating human action may not be able to move the scanner fast enough to feed the barcode, which worsens the effective utilization of the application and its battery power. The issue of energy consumption does not limit itself to battery-powered embedded devices, but also to embedded systems that remain running for long periods of time on stationary power. A scheduling display system in a train station and a household intruder alarm system are two examples of such embedded systems.

A recent survey shows that the Information and Communication Technologies (ICT) sector has a carbon footprint equal to the aviation industry, accounting for 3–4% of the world's carbon emissions [7]. Take the above stocktaking scenario as an example. If there are only 100 such devices used on Hong Kong Island and each device can save merely 1 Watt, according to the calculation provided by National Semiconductor [13] as shown in Figure 1, it helps save 1666 pounds of CO_2 emissions per year.

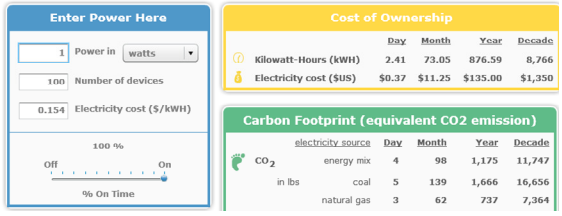

Figure 1. CO_2 emissions from 100 scanners (according to [13]).

To this end, in this paper, we address the problem by utilizing the dynamic voltage and frequency scaling (DVFS) that comes with many modern processors such as Intel's Enhanced Intel Speedstep® Technology [9]. The general idea is a well-received concept: By lowering the processor frequency and voltage dynamically, the workload of an application could be spread over a period of time in return for energy savings. For CPU-bound tasks such as decoding barcode strips into readable characters, we define workload as the number of processor cycles required to complete the task.

The relationship between power consumption (P), voltage (V), and frequency (f) can be estimated as follows [23]:

$$P \propto V^2 f \quad (1)$$

$$V \propto f \quad (2)$$

* This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project nos. 111107, 123207, and 717308).

† All correspondence should be addressed to Dr. W. K. Chan at Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. Tel: (+852) 2788 9684. Fax: (+852) 2788 8614. Email: wkchan@cs.cityu.edu.hk.

In other words, as frequency is reduced linearly, voltage is also reduced linearly in a predetermined, hardware-specific fashion. This will result in energy savings in approximately cubic order and only gradual performance degradation.

Reducing voltage and frequency does not necessarily reduce power consumption owing to the longer time it takes to complete a task. In our study, we model the problem as a constrained optimization problem in finding the CPU frequencies that minimize the overall energy consumption for all tasks, with task deadlines as constraints. Once the optimal frequencies are computed, the processor is set to run at the computed frequency (and the corresponding stable voltage) during the execution of each task. The problem is challenging in practice because the underlying processors only support a discrete number of frequencies, which makes the problem NP-hard [21][22]. It is unlikely that there exists an efficient algorithm that solves the problem in the polynomial time [6]. Consequently, like many other researchers, we first assume a continuous spectrum of processor speeds being available. After computing the optimal frequency f_i for a task T_i over a time period t_i, we attempt to simulate the execution using the lower adjacent frequency f_A and the higher adjacent frequency f_B over time periods t_A and t_B such that

$$f_i t_i = f_A t_A + f_B t_B$$
$$t_A + t_B \le t_i$$
$$E(f_A) + E(f_B) \le E(f_i)$$

where $E(f_i)$ denotes the processor energy computation when the processor is running at frequency f_i. Chen et al [3] showed in their experiment that the above linear combination achieves lower energy consumption than running at f_B over the entire duration t_i.

As pointed out in recent research [2][4][15], it is important not to overlook the energy consumptions of other system components such as memory, I/O devices, and basic circuitry. In our study, we model system-wide energy consumption by decomposing power usage and execution times into frequency-dependent and frequency-independent variables to reflect on-chip (i.e., CPU) and off-chip (i.e., other components) computations respectively.

In this paper, we consider a novel extension to the traditional definition of deadlines by allowing a finite number of possible "deadlines" for each task. To clarify this point, let us consider a simplistic version of the above-mentioned example with barcode scanner:

TABLE 1. EXAMPLE OF BARCODE SCANNER

Case	Prob	Full-Speed Time (ms)	Deadline (ms)
1	95%	500	800
2	5%	1500	2000

We suppose that the scanner can read a barcode in 500 ms for 95% of the time (case 1), and takes 1500 ms at the worst case for the remaining 5% (case 2). Moreover, we further assume that the entire duration is dominant by the CPU component (i.e., the on-chip component), and that each barcode scanning should not take more than 2 s (or 2000 ms). Applying our approach, two "target deadlines" may be defined for this scanning task: first at 800 ms and the second at 2000 ms. Our frequency assignment algorithm first takes the value of 800 ms with respect to the offset of the task as the target deadline for the task and computes the optimal frequencies. The idea is that since 95% of all executions complete in 500 ms at full speed, energy savings by reducing processor frequency is already possible with the 800 ms deadline. Performance deterioration can also be controlled to within 800–500=300 ms. If the first target is missed, the algorithm then takes 2000 ms as the deadline and computes another set of frequencies. In this case, the execution is assumed to behave as in the worst-case scenario, and the goal of the algorithm now is to fulfill the second deadline while still attempting to conserve energy.

While the worst-case deadline is usually defined based on application-specific requirements, definition of other target deadlines does not necessarily involve much additional efforts because these can be inferred from the operational profiles of the tasks by adding a fixed percentage of the execution time as slacks. Referring to the example above, we add 60% slacks to the usual execution time of 500 ms as the corresponding target deadline of 800 ms. The fixed percentage can be applied to all operational profiles to infer multiple target deadlines provided that the computed deadlines do not exceed the worst-case deadline (2000 ms in the above example).

The main contribution of this paper is twofold: First, we develop a multi-deadline frequency assignment strategy that supports various performance levels for *most* executions of each task and attempts to enforce the task deadline with respect to the worst-case execution scenario. Our strategy is able to minimize the energy consumption in either case. Second, we report a simulation study on our strategy. The experiment result shows that with the presence of 150% slack time, the two versions of our algorithms achieve 52% energy savings on average compared to without any power management scheme, and up to 72% savings if all execution times are accurately predicted.

For the rest of the paper, unless otherwise noted, we denote the set of execution times and deadlines for each task as *prediction cases* or *test cases*, and a selected deadline for a task at any instant of the algorithm execution as a *deadline*. A test case *holds* if it correctly predicts the timing of an execution.

The rest of the paper is organized as follows: Section II summarizes the related research. Section III formulates our optimization problem. Section IV describes our algorithm and its variations. Section V presents the empirical results and our evaluation. Section VI concludes the paper.

II. RELATED WORK

A. Power-Aware Frequency Assignments

A number of earlier studies focus on processor power savings without considering energy consumed by other off-chip system components [8][19]. Existing approaches can be classified into three categories: interval, inter-task, and intra-task.

Weiser et al adopted an interval approach in [19] by monitoring CPU utilization at regular intervals. Based on the statistics gathered, the clock frequency and voltage are reduced

whenever the utilization drops below a predefined threshold. On the other hand, the CPU is accelerated again if the utilization percentage exceeds a certain threshold. The idea is to minimize energy consumption by reducing the amount of idle time in serving the same number of requests. Because the algorithm only utilizes data from the preceding round of the task execution, its prediction of future CPU utilization can be inaccurate, resulting in a suboptimal frequency assignments.

Inter-task approach works at a finer granularity by assigning different frequencies per task rather than per interval. Shin and Choi [17] considered a modified scheduler for fixed priority scheduling in hard real-time systems. Their approach aims at lowering the frequency during the intervals in between executions of different tasks when the CPU is idle. CPU frequency and voltage of the active task is reduced whenever there is no task pending for immediate execution (i.e., in the run queue). If the CPU is predicted to be idle for sufficiently long time, the system enters power-down mode. Shin and Choi incorporated the rate of change of processor speed into their calculation of the optimal frequency. However, due to the expensive computation involved, they resorted to a heuristic solution that disregarded this factor in return for less overhead in the scheduling algorithm. The algorithm works well experimentally across several subject applications, and achieves better energy savings than the selected interval techniques despite that its accuracy is subject to the duration in between speed changes. If the processor speed changes frequently, the simplified heuristics in the algorithm may not harvest all potential power savings. Our approach can be classified as an inter-task as we compute distinct frequency per task. However our approach is different from [17] in that we utilize operational profiles of the tasks to compute CPU frequencies instead of referring to the current workload. Unlike [17], in our approach, the rate of change of processor speed does not depend on whether the run queue is empty.

Intra-task energy optimization allows a task to be run at different frequencies throughout its execution period. Unlike the inter-task strategies, the intra-task approach may require additional information of each task at the design time or at the run-time (e.g., function/method of the high-level application being executed on a virtual machine) to determine its optimal frequency at any point in time. An example of this approach is presented by Rauch et al [14] where the Java Virtual Machine (JVM) is used to profile the CPU, memory, and I/O access of an application. The statistics collected in the execution context are checked by a separate thread at regular intervals and the processor frequency is changed if the application exhibits a high degree of off-chip activities in the past interval. Therefore their approach can be classified as a hybrid intra-task/interval approach where statistics of a task is collected alongside its execution in the JVM, and CPU frequency is adjusted at regular intervals. Implementing energy saving algorithms at the JVM level is beneficial in the sense that more programming constructs are available for considerations by a frequency assignment algorithm compared to relying solely on an operating system. The proposed algorithm is implemented as a standard Java interface, which targets it to be platform-independent. Conversely, the implementation is limited to applications where runtime instrumentation is possible (such as those written in Java), and imposes 2–6% instrumentation overhead by inserting profiling codes within the execution context of applications monitored. Our approach can also be classified as intra-task in the case if an initial test case fails to predict the completion of a task (after 800 ms in our previous example in Section I), our algorithm computes the frequency assignment for the remaining execution of this task and assumes that the next test case holds under the newly computed frequency assignment.

Variable deadlines have been proposed and studied by Shih and Liu [16]. They considered the case when deadlines of tasks can be constantly changing during their executions. Their approach models the deadline as a random process, and utilizes historical data sampling and simulations to construct probability distribution functions for different elapsed times since the first arrival of a task. A requirement engine is introduced to track changes in timing requirements for the underlying scheduling algorithm. Although our approach is similar in that each task may be associated with different deadlines at different times, our approach focuses on finding optimal frequency assignments with predefined deadlines, while Shih and Liu focus on how to gather updated deadlines without providing a concrete implementation of frequency assignment.

B. *Virtual Machine Instrumentation and Profiling*

It is important to mention virtual machine instrumentation because it can help automate the test case generation process. Similar to [10], [14], and many other studies, our approach relies on execution timing and deadlines (namely test cases as we define in Section I) which can be programmatically collected by means of instrumentation without the need to change the application source code. For instance, functions and methods of a program written in a high-level programming language can be instrumented for timing prediction of normal and worst-case executions. Deadlines for normal test cases can also be deduced manually or automatically from these data. Wilhelm et al [20] presented an overview of the methodology and tools available to determine worst-case execution times for real-time tasks. Although in this paper we assume that we are given the test cases, the analysis and automation tools described above show that gathering the execution times required in our approach is technically feasible.

III. OUR MODEL

In this section, we formulate our model formally.

A. *Worst-Case Execution Time*

Consider a set of independent tasks $\{T_1, \ldots, T_n\}$. To compute the energy consumption of a task T_i, we need to examine the effect of processor frequency on the execution time. Following the previous work [4], we define the worst-case execution time (WCET) as the longest time to complete a task at the full processor speed. In the presence of off-chip computations, similar to [4], we further decompose the WCET of each task execution into an on-chip component and an off-chip component. Assuming a single processor system, the WCET w_i of task T_i is:

$$w_i = w_i^{on} + w_i^{off}$$

where w_i^{on} is the execution time on-chip and is *dependent* on CPU frequency; w_i^{off} is the execution time off-chip and is *independent* of CPU frequency. To reflect the change in CPU frequency on the overall execution time, we assume that at a lower frequency, the on-chip component takes proportionally longer period (i.e., $\frac{w_i^{on}}{f}$) to do the same amount of work in terms of number of CPU cycles. The execution time $t_i(f)$ of task T_i is:

$$t_i(f) = \frac{w_i^{on}}{f} + w_i^{off}, \quad f \in (0,1] \tag{3}$$

where f is frequency normalized to 1 when the CPU is at its maximum speed.

B. Test Cases and Deadlines

We define test cases τ_{ij} for task T_i as triples sorted in ascending order of target deadlines:

$$\begin{aligned} &\tau_{ij} = (t_{ij}^{on}, t_{ij}^{off}, D_{ij}), && 1 \le i \le n,\ 1 \le j \le |\tau_i| \\ &\tau_i^* = (t_i^{*on}, t_i^{*off}, D_i^*), && 1 \le i \le n \\ &D_{ij} \le D_{ik}, && \text{if } 1 \le j < k \le |\tau_i| \end{aligned} \tag{4}$$

where t_{ij}^{on}, t_{ij}^{off}, and D_{ij} are the on-chip, off-chip execution times, and deadline of each test case respectively; $|\tau_i|$ is the number of test cases for the task; and τ_i^* is the test case selected by the algorithm for T_i. Note that if there is any task that cannot meet its deadline even when the processor is running at full speed, we will refer to the default rule of arbitration (i.e., reject the task or simply run at full speed).

We recall that the purpose of allowing multiple test cases for a task is to sustain the performance for execution scenarios that are more likely to occur than the worst-case scenario, it follows that for each task T_i,

$$\begin{aligned} &P(\tau_{ij}) \ge P(\tau_{ik}), && \text{if } 1 \le j < k \le c_i \\ &\sum_{j=1}^{c_i} P(\tau_{ij}) \le 1 && \text{for all } i \end{aligned} \tag{5}$$

where $P(\tau_{ij})$ is the probability that τ_{ij} holds for task T_i.

C. Energy Model

We adopt the system-wide energy model presented in [22] and [23], which also takes into account the off-chip and on-chip power consumptions. For any unit of time (t_A) spent performing task T_i, power is consumed by the following active components during the execution of T_i: frequency-sensitive components (denoted as $P_i^{on}(f)$), frequency-insensitive components that can be put into sleep modes when not running T_i (denoted as P_i^{off}), and other components that consume static power during the execution of T_i (denoted as P_S). Sensitivity to frequency is defined as whether a component consumes different amounts of energy when the corresponding CPU frequency is changed. Following equations (1) and (2), we model energy utilization as:

$$\begin{aligned} E_i(f) &= P_s\, t_A + (P_i^{off} + P_i^{on}(f))\, t_i(f) \\ &= P_s\, t_A + (P_i^{off} + C_i f^3))\, t_i(f) \end{aligned} \tag{6}$$

where t_A is the time period allocated to T_i; C_i is the task-specific effective capacitance being switched per clock cycle. Note that we refer to equation (3) in the context of WCET and expand $t_i(f)$ to w_i^{on} and w_i^{off} in equations (7) and (8) below. These terms can be replaced with t_i^{*on} and t_i^{*off} in the context of test cases. Taking the first derivative, we have:

$$E_i'(f) = 3C_i\, w_i^{off} f^2 + 2C_i w_i^{on} f - P_i^{off} w_i^{on} f^{-2} \tag{7}$$

We further take the second derivative:

$$E_i''(f) = 6C_i w_i^{off} f + 2C_i w_i^{on} + 2P_i^{off} w_i^{on} f^{-3} \tag{8}$$

For $f, w_i > 0, P_i^{off} \ge 0$, we have equation (8) > 0, which shows that equation (6) is convex.

We can set equation (7) to 0 and solve the quartic equation for f analytically. It has been shown that solving v quartic equations can be achieved in $O(v^3)$ time [12]. For the special case where there is no off-chip time, reference [22] presents a close formula for solving the optimal frequency f^*. We define f_i^* as the maximum of f^* considering only task T_i, and f_{min}, the lowest frequency supported by the processor normalized to 1. In general, since dom(E_i) is the set of all positive real numbers which is also convex, our problem can be formulated into a convex optimization problem with constraints.

D. Convex Optimization Problem

Given the above model, we formulate our optimization problem as follows:

$$\min \sum_i E_i(f_i)$$

$$\text{s.t.} \sum_i \left(\frac{t_i^{*on}}{f_i} + t_i^{*off}\right) \le D_i^*, \text{ for i = 1,..., n} \tag{9}$$

$$f_i^* \le f_i \le 1,\ 0 < f_i^* \le 1 \tag{10}$$

Constraint (9) ensures that each task is completed before its selected deadline. Since the objective function is a summation of convex functions of f_i as discussed in the previous subsection, the optimization problem is also convex.

IV. Algorithm

A. Preprocessing of Test Cases

Constraints (4) and (5) specify the rules when defining test cases. It is worthy to note that real-life executions of tasks may not follow these constraints. We illustrate our algorithm using the hypothetical execution times of an arbitrary task T_i as shown in Table 2. First of all, notice that τ_{i2} and τ_{i3} are nearly identical. We can combine them into τ_{i23} = (10, 0, 20) with probability 0.2 in order to reduce the incurred computation overhead. To satisfy constraint (4), we first sort the test cases by deadlines in ascending order: $\{\tau_{i23}, \tau_{i5}, \tau_{i4}, \tau_{i1}\}$. However, this sequence violates constraint (5) because $P(\tau_{i23}) < P(\tau_{i5})$. In this

case, we remove τ_{i23} from the test suite. If further reduction of the test suite size is required owing to practical limitations, test cases may be combined further to achieve this goal. For instance if only 2 test cases are allowed per task, one may combine τ_{i2}, τ_{i3}, τ_{i4}, and τ_{i5} into a new test case with the weighted average of the execution times. For the rest of the discussion, we assume the given test cases are preprocessed and follow these constraints. It will be interesting to investigate as a separate study whether automatic preprocessing can be done effectively.

TABLE 2. EXAMPLE OF TEST CASE PREPROCESSING

Case	$P(\tau_{ij})$	t_{ij}^{on}	t_{ij}^{off}	D_{ij}
τ_{i1}	0.1	40	0	100
τ_{i2}	0.1	10.001	0	20
τ_{i3}	0.1	10	0	20
τ_{i4}	0.3	30	0	60
τ_{i5}	0.4	20	0	40

Preprocessing test sets may reduce the number of test cases per task but we can never be definite which test case will most accurately describe the on-chip and off-chip execution times of a task. To ensure that deadlines are met in all cases, a simple solution is to consider all test cases in constraint (9). However this prevents maximum energy savings since additional constraints that do not accurately predict task executions are included into the computation. In this paper, we present our heuristic Test-Guided Power Management (TGPM) algorithms to tackle this problem.

B. TGPM-ALL Algorithm

The following is the baseline version of our algorithm:

Algorithm 1. TGPM-ALL

```
1:  Q ← insert new test cases into queue sorted according to constraint (4);
2:  T_guess ← get earliest test case for each task from Q;
3:  compute f* for each task in T_guess;
4:  f ← call barrier(T_guess, f*) to find optimal frequencies;
5:  update f for each task;
6:  T ← Task for T_guess(1);
7:  run checker(T) at the expected or actual completion of T;

8:  procedure checker(T)
9:    if T_guess(1) is for task T then remove it from T_guess;
10:   if T is actually completed then remove all test cases of T from Q;
11:   else
12:     remove T_guess(1) and unfeasible test cases of T from Q;
13:     if no more test case of T exists in Q then
14:       run at full speed and run checker(T) on completion of T;
15:       return;
16:     end if;
17:     T_guess ← insert earliest test case for T from Q;
18:     repeat 4–5;
19:   end if;
20:   repeat 6–7;
21: end procedure
```

In lines 1–2, we assume Earliest Deadline First (EDF) scheduling [18], sort all test cases for all tasks, and put them in Q. For each task T_i, we pick the first test case as τ_i^* and store them in T_guess. In lines 3–4, the chosen test cases and constraints are passed into the Interior Point algorithm to compute an optimal set of frequencies. The algorithm is described in details in the next subsection. In line 5, the computed frequencies are enforced. In line 6–7, the expected completion time of the first task is computed and **checker** is scheduled to run at that time or when the task actually completes.

In lines 9–10 if the task is actually completed, all test cases of the task is removed in line 11. Otherwise, the task does not complete as expected. Lines 12–13 remove all unfeasible test cases from Q and checks if there is another feasible test case for this task. A test case is feasible if the actual elapsed on-chip and off-chip times are smaller than or equal to those of the test case. If no such test case exists, line 14 runs the arbitration rule for missing deadlines. Otherwise, lines 17–20 put the next test case into T_guess, and the algorithm retrieves the task with the earliest deadline from T_guess and repeats itself.

C. Interior Point and Infeasible-Start Newton Methods

To solve the constrained optimization problem described by equation (9), we employ the Interior Point Method with indicator function $\phi(f)$, and the infeasible start Newton method [1]. These algorithms run in polynomial time and are well studied in the field of Convex Optimization. Following the literature in [1], $\phi(f)$ can be approximated using a logarithmic barrier function

$$\phi(f) = -\frac{1}{t}\sum_{\mathrm{i}} \log\left(D_i - \frac{w_i^{on}}{f_i} - w_i^{off}\right) - \frac{1}{t}\log(f_i - f_i^*) \quad , t \to \infty$$
$$-\frac{1}{t}\log(1^* - f_i)$$

We let 1* denote a real number slightly larger than 1 (we use 1.000001 in our experiment). This is to overcome the non-zero input domain of the log function. The gradient and Hessian terms for the problem described by (9) are as follows:

$$\nabla\phi = \left(\frac{\partial\phi}{\partial f_i}\right) = \left(-\frac{w_i^{on}}{f_i^2(D_i - w)} + \frac{1}{1^* - f_i} - \frac{1}{f_i - f_i^*}\right) \tag{11}$$

$$\nabla^2\phi = \left[\frac{\partial^2\phi}{\partial f_i\,\partial f_j}\right] = \begin{cases} \dfrac{w_i^{on} f_i^2 w_j^{on}}{f_i^4 (D_i - w)^2 f_j^2} & ,i > j \\ \dfrac{w_i^{on}(2f_i(D_i - w) + w_i^{on})}{f_i^4 (D_i - w)^2} + & ,i = j \\ \dfrac{1}{(1^* - f_i)^2} + \dfrac{1}{(f_i - f_i^*)^2} & \\ 0 & ,i < j \end{cases} \tag{12}$$

$$w = \sum_{j=1}^{i} \frac{w_j^{on}}{f_j} + w_j^{off} \tag{13}$$

We implement the infeasible-start Newton algorithm by making use of equations (11) to (13). The optimization algorithm is outlined below. Initializations of parameters in lines 1–2 are typical values suggested from empirical studies. We start the algorithm with a feasible input of normalized frequencies of 1 (i.e., full CPU speed). Lines 2–4 narrow down the range of the optimal values for the input vector f. Lines

4–10 are the implementation of Newton method. Specifically, lines 7–8 performs backtracking line search by checking whether the Euclidean norm of the gradient in equation (11) is sufficiently small before updating the vector *f*. Note that the feasibility check in line 7 ensures that all values in the vector *f#* satisfy constraint (10).

Algorithm 2. Interior Point Method (barrier)

```
1:  initialize: ε = 0.000001, t = 1, α = 1/3, β = 5/6, μ = 2, and strictly
    feasible f = 1 and f# = 1;
2:  while 3|T|/t ≥ ε do
3:    norm_r ← Compute ∇φ according to equation 11;
4:    while norm_r > ε and f <> f# do
5:      t ← 1;
6:      Δf ← Compute ∇²φ according to equation 12 and 13;
7:      while norm(∇φ) > (1-α*t)*norm_r or f# not feasible
8:        t ← β*t, f# ← f+t*Δf, compute ∇φ according to equation 11;
9:      f ← f#, norm_r ← norm(∇φ);
10:   end do;
11: end do
```

D. Asymptotic Complexity

The overall complexity of TGPM-ALL depends on a number of parameters, namely n and $c = \sum_{i=1}^{n} |\tau_i|$, the total number of test cases. Lines 1, 2, and 17 of Algorithm 1 have complexity O(log(*c*)). Line 3 involves solving n quartic equations, which can be achieved in $O(n^3)$ as described previously. Line 4 converges in exactly $b = \left\lceil \frac{\log(3|T|/\varepsilon)}{\log \mu} \right\rceil \left(\frac{E_{NPM} - E^*}{\gamma} + 6 \right)$ iterations [1] where E_{NPM} is the energy consumed when there is no power management (i.e., processor at full speed); E^* is the optimal energy consumption; and γ is the upper bound on the component values of Δf between two successive Newton iterations (line 4 of Algorithm 2). In the worst case, the **barrier** method is called *c* times. The overall complexity of TGPM-ALL is, therefore, $O(n^3 + \log(c) + cb)$, or $O(n^2 + (\log(c) + cb) / n)$ for each task.

E. TGPM-N Algorithm

The number of calls to the **barrier** method may become overwhelming when the total number of test cases is reasonably large. To address this problem, we present an enhanced version TGPM-N based on TGPM-ALL:

Algorithm 3. TGPM-N

```
1–7:  (same as TGPM-ALL)
8:    k(Ti) ← 0 for all tasks Ti;
      procedure checker(T)
9:      if T_guess(1) is for task T then remove it from T_guess;
10:     if T is actually expected then remove all test cases of T from Q;
11:     else
12:       k(T) ← k(T)+1;
13:       remove T_guess(1) and unfeasible test cases of T from Q;
14:       if no more test case of T exists in Q or k(T)=N then
15:         remove all test cases of T from Q;
16:         run at full speed and run checker(T) on completion of T;
17:         return;
18:       end if;
19:     repeat 6–7;
20:   end procedure
```

Line 8 initializes *k* to 0 for all tasks. Line 12 keeps track of the number of failed test case attempts for task *T*. If it meets the predefined value *N*, lines 15–16 discard remaining test cases for the task and execute the task at full speed until completion. The overall complexity is then reduced to $O(n^2+\log(c)/n+Nb)$ for each task.

V. Evaluation

We implemented TGPM-ALL and TGPM-1 in MATLAB and compared their performance in terms of energy savings. Owing to the potentially large number of tasks and test cases, the algorithm must also run efficiently and should not incur too much overhead on the system. Hence, we also instrumented the two versions to report the number of Newton iterations executed in line 4 of Algorithm 2. In our evaluation, we selected the BEST algorithm as the one that always correctly identified the test case that best described each task. Although it is not a plausible algorithm unless we have knowledge into the future as to how all tasks will execute, it nevertheless serves as a theoretical bound of our strategy and a reference for comparison.

A. Experimental Setup

Without loss of generality, we let $P_s\ t_A$ in equation (6) be 0, as this factor does not depend on the variables (time or frequency) manipulated by our algorithm.

For the number of tasks n, we experimented with 10, 20, and 50 simultaneous tasks for the optimization problem and find that for the evaluation criteria stated above, their statistical patterns agree with each other. Therefore in the following discussions, we only present the results with 20 simultaneous tasks. Note that each data point in our figures represents average values of 50 trial runs.

We evaluate the effectiveness of our approach using the baseline energy consumption E_{NPM} when all tasks are running at full speed. The reported energy consumptions are normalized to $E_{NPM.}$ First we study the effects of energy savings with respect to test suite sizes by setting $\frac{P^{off}}{P^{on}(f)} = 0.05$, $C = 1$, $t_i^{*off} = 0$ and $f_{min} = 0.2$. (We note that the effects of these parameters have also been studied by Zhu and Aydin [22].) We summarize the findings in Section V-B.

In our simulation, we randomly generate t_i^{*on} from 0.1 to 1.0 based on uniform distribution. To guarantee achievable deadlines, we define $S > 0$ as the amount of slack time between task completion at full speed and the deadline. *S* is expressed as a multiple of the full speed execution time. We let $S = 1.5$ to ensure sufficient slack time while we study the effects of test suite sizes. After generating all test cases, we randomly pick one test case for each task as the true runtime characteristic of the task.

B. Effects of P^{off}, C, and t_i^{*off}

It has been shown in [22] that the effects of these parameters are similar on all DVFS-based power management schemes. As $\frac{P^{off}}{P^{on}(f)}$ increases, off-chip components consume relatively more energy. Intuitively this has an adverse effect on all processor-based power management schemes as less power

can be saved on-chip relatively by frequency and voltage manipulation. Conversely, increased switching capacitance (C) and off-chip workload reduce energy-efficient frequencies, and will therefore benefit all DVFS power schemes.

C. Effects of Test Case Size on Energy Consumption

Figure 2 shows the effects of test suite size on energy savings. As expected, TGPM-ALL performs better than TGPM-1 (denoted by ONCE) in the simulation. TGPM-ALL behaves much closer to the hypothetical BEST algorithm. From Figure 2 with as many as 20 simultaneous tasks and 10 test cases per task, it can still save up to 60% of CPU power compared to the case when no power management scheme is active, although it slowly deteriorates as the number of test cases and failed test cases increases. Note that the performance of BEST is unaffected by test suite size since we assume that it always picks the correct test case regardless of test suite size.

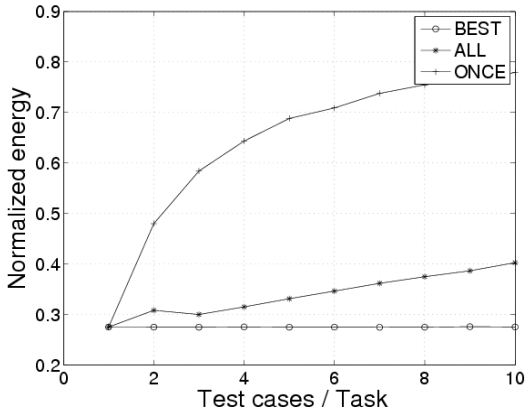

Figure 2. Effects of test suite size on energy consumption.

D. Effects of Slacks on Energy Consumption

Another important factor that affects energy savings is the amount of available slacks. Figure 3 shows that all three algorithms achieve more energy savings as the available slacks increase. Note that TGPM-ALL and BEST have very similar sensitivity to available slacks. As S increases from 0.1 to 1.5 times of the full speed execution time, both can achieve additional energy savings of about 54% (from 17% to 71%). On the other hands, TGPM-1's energy savings only increases by 29% (from 14% to 43%) for the same increase of available slacks. This can be explained by the fact that TGPM-1 only benefits from the additional slacks of the first test case, and then switches to full speed immediately if the case fails.

To put the energy savings in the context of CO_2 emissions, let us consider the amount of CO_2 emissions generated by the ICT sector, which was 3.5% [7] of total global emissions as of 2006 [11]. Suppose 1% of the emissions is related to embedded systems with devices capable of implementing our proposed algorithm TGPM-1 with an average of 10% slack time in deadlines. The reduction of CO_2 emissions will be roughly equivalent to the emission of 255,437 average cars commuting for one year [5].

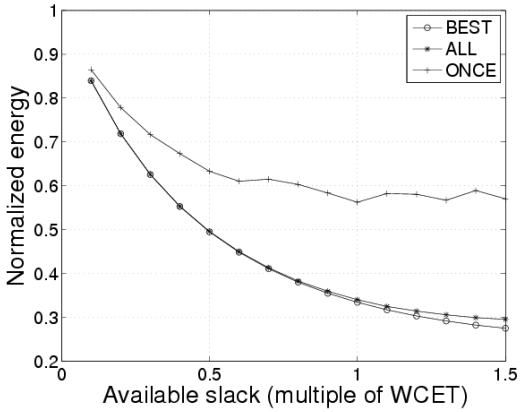

Figure 3. Effects of slacks on energy consumption.

E. Effects of Test Case Size on Efficiency

In the previous criteria, TGPM-ALL is shown to be effective in energy savings. Figure 3 shows that TGPM-1 is more efficient and is comparable to the BEST algorithm. The runtime of the algorithm increases linearly for TGPM-ALL; whereas the other two algorithms are unaffected by test suite size. It is also interesting to note that the position of the best test case for each task also plays a role in the efficiency of TGPM-ALL. In Figure 4, we include two scenarios: ALL (worst) and ALL (random). ALL (worst) always assigns the test case with the latest deadline as the correct test case of each task. ALL (random), on the other hand, randomly assigns the correct test case for each task in the simulation. We see that the increase of execution time in the scenario of ALL (worst) is noticeably faster than in the scenario of ALL (random). This reflects the performance / energy tradeoff for both the tasks and the frequency assignment algorithm itself. TGPM-1 ensures shorter completion of tasks and efficient frequency assignment by switching to full speed after the most probable test case fails to hold.

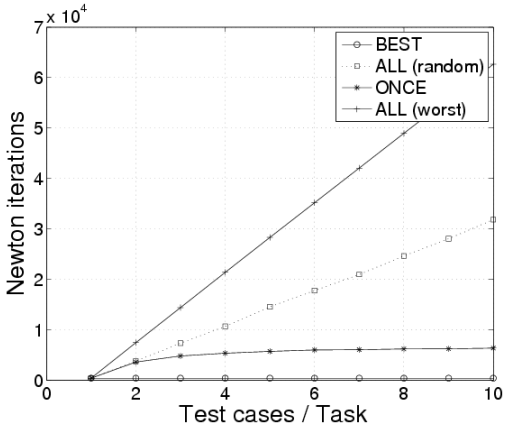

Figure 4. Effects of test suite size on efficiency.

There are a couple of threats to validity about the experiment. First, owing to the use of synthetic test cases, our simulation model cannot accurately model the fact that the first test case for each task always has the highest probability of predicting the actual execution times than remaining cases. It may be to the advantage of TGPM-1 if we consider overall energy

consumption in the long run with probabilities. Second, we only compare one instance of TGPM-N in this paper. The results of other instances are uncertain. However, we believe that they tend to lie between TGPM-ALL and TGPM-1.

VI. Conclusion

Energy efficiency of embedded systems is becoming more important owing to environmental issues. Most processors today support DVFS, which allows the scaling down of CPU voltage and frequency to save energy. Many proposed frequency assignment strategies only consider minimization of power consumption and meeting real-time task deadlines. We have presented a heuristic algorithm to handle the frequency assignment problem for embedded systems with multiple soft deadlines. We have modeled the problem as a convex optimization problem and utilized the Interior Point method in our algorithm to solve for optimal frequencies. To allow flexibility in maintaining performance, our approach accepts multiple target deadlines for each task. We have developed the TGPM algorithms to try all or part of the supplied test cases. We also reported an experiment on the performance and efficiency aspect of our MATLAB implementation. The empirical results show that the TGPM algorithms can be effective in leveraging performance and energy savings.

There are various future directions to explore. First, it will be interesting to study the effects of multiple deadlines with a real execution on an embedded system possibly with multi-core processors. Careful selection and filtering of test cases can significantly improve the performance of TGPM. Automation of this process will simplify and contribute to better energy savings. Another direction is to explore more efficient algorithms to solve the optimization problem.

References

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[2] D. J. Brown and C. Reams. Toward energy-efficient computing. *ACM Queue*, 8 (2): 30–43, 2010.

[3] B. Chen, W. P. T. Ma, Y. Tan, A. Fedorova, and G. Mori. GreenRT: a framework for the design of power-aware soft real-time applications. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture* (*in conjunction with the 35th International Symposium on Computer Architecture* (*ISCA-35*)). Beijing, China, 2008.

[4] K. Choi, W. Lee, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD 2004*), pages 29–34. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[5] *Drive Smart Calculator*. Available at http://www.drivesmartsavegreen.com/calculator.html. Last access March 2010.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1990.

[7] Global Action Plan. *Inefficient ICT Sector's Carbon Emissions Set to Surpass Aviation Industry: December 2007*. Available at http://globalactionplan.org.uk/first-national-survey-reveals-60-businesses-are-lacking-support-sustainable-ict-strategies-december-.

[8] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the IEEE Real-Time Systems Symposium* (*RTSS 1998*), pages 178–187. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[9] Intel Corporation. *Processors: Frequently asked questions for Intel Speedstep Technology*. Available at http://www.intel.com/support/processors/sb/CS-028855.htm. Last access March 2010.

[10] S. Liu, Q. Wu, and Q. Qiu. An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems. In *Proceedings of the 46th Annual Design Automation Conference* (*DAC 2009*), pages 782–787. ACM Press, New York, NY, 2009.

[11] Millennium Development Goals Indicators. *Carbon Dioxide Emissions* (*CO2*), *Thousand Metric Tons of CO2* (*CDIAC*). 2009. Available at http://mdgs.un.org/unsd/mdg/SeriesDetail.aspx?srid=749&crid=.

[12] C. Moler. Cleve's corner: roots — of polynomials, that is. *The MathWorks Newsletter*, 5 (1): 8–9, 1991.

[13] National Semicondutor. *CO2 Calculator*. Available at http://www.national.com/analog/powerwise/co2_calculator. Last access March 2010.

[14] M. Rauch, A. Gal, and M. Franz. Dynamic adaptive power management for — and by — a Java virtual machine. Technical Report No. 06-19. Donald Bren School of Information and Computer Science, University of California, Irvine, Irvine, CA, 2006.

[15] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems*, 5 (1): 200–224, 2006.

[16] C.-S. Shih and J.W.S. Liu. Acquiring and incorporating state-dependent timing requirements. *Requirements Engineering*, 9 (2): 121–131, 2004.

[17] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference* (*DAC 1999*), pages 134–139. ACM Press, New York, NY, 1999.

[18] J. A. Stankovic, K. Ramamritham, and M. Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, Norwell, MA, 1998.

[19] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation* (*OSDI 1994*), page Article No. 2. USENIX Association, Berkeley, CA, 1994.

[20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom. The worst-case execution-time problem: overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7 (3): Article No. 36, 2008.

[21] X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: lower bound and approximation. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD 2006*), pages 516–521. ACM Press, New York, NY, 2006.

[22] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD 2006*), pages 528–534. ACM Press, New York, NY, 2006.

[23] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD 2004*), pages 35–40. IEEE Computer Society Press, Los Alamitos, CA, 2004.