# Quasi-Static Assignment of Voltages and Optional Cycles in Imprecise-Computation Systems With Energy Considerations

Luis Alejandro Cortés, *Member, IEEE*, Petru Eles, *Member, IEEE*, and Zebo Peng, *Senior Member, IEEE*

***Abstract*—For some realtime systems, it is possible to tradeoff precision for timeliness. For such systems, typically considered under the imprecise computation model, a function assigns reward to the application depending on the amount of computation allotted to it. Also, these systems often have stringent energy constraints since many such applications run on battery powered devices. We address in this paper, the problem of maximizing rewards for imprecise computation systems that have energy constraints, more specifically, the problem of determining the voltage at which each task runs as well as the number of optional cycles such that the total reward is maximal while time and energy constraints are satisfied. We propose a quasi-static approach that is able to exploit, with low online overhead, the dynamic slack that arises from variations in the actual number of task execution cycles. In our quasi-static approach, the problem is solved in two steps: first, at design-time, a set of voltage/optional-cycles assignments are computed and stored (offline phase); second, the selection among the precomputed assignments is left for runtime, based on actual completion times and consumed energy (online phase). The advantages of the approach are demonstrated through numerous experiments with both synthetic examples and a real life application.**

***Index Terms*—Energy management, imprecise computation, quasi-static, realtime.**

## I. Introduction

THERE exists several application areas, such as image processing and multimedia, in which approximate, but timely, results are acceptable. For example, fuzzy images in time are often preferable to perfect images too late. In these cases it is, thus, possible to tradeoff precision for timeliness. Such systems have been studied in the frame of imprecise computation (IC) techniques [23], [14]. These techniques assume that tasks are composed of mandatory and optional parts: both parts must be finished by the deadline but the optional part can be left incomplete at the expense of the quality of results. There is a function associated with each task that assigns a reward as a function of the amount of computation allotted to the optional part: the more the optional part executes, the more reward it produces.

Also, power and energy consumption have become very important design considerations for embedded systems, in particular for battery powered devices with stringent energy constraints. The availability of vast computational capabilities at low cost has promoted the use of embedded systems in a wide variety of application areas where power and energy consumption play an important role.

The tradeoff between energy consumption and performance has extensively been studied under the framework of dynamic voltage scaling (DVS): by lowering the supply voltage quadratic savings in dynamic energy consumption can be achieved, while the performance is approximately degraded in linear fashion. One of the earliest papers in this area is by Yao *et al.* [29], where the case of a single processor with continuous voltage scaling is addressed. The discrete voltage selection for minimizing energy consumption in monoprocessor systems was formulated as an integer linear programming (ILP) problem by Okuma *et al.* [19]. DVS techniques have been applied to distributed systems [9], [12], [15], and even considering overheads caused by voltage transitions [31] and leakage energy [1]. DVS has also been considered under fixed [25] and dynamic priority assignments [13].

While DVS techniques have mostly been studied in the context of hard realtime systems, IC approaches have, until now, disregarded the power/energy aspects. Rusu *et al.* [20] recently proposed the first approach in which energy, reward, and deadlines are considered under a unified framework. The goal of the approach is to maximize the reward without exceeding deadlines or the available energy. This approach, however, statically solves at compile-time the optimization problem and, therefore, considers only worst cases, which leads to overly pessimistic results. A similar approach (with similar drawbacks) for maximizing the total reward subject to energy constraints, but considering that tasks have fixed priority, was presented in [30]. Such approaches can only exploit the static slack, which is due to the fact that at nominal voltage the processor runs faster than needed.

Most of the techniques proposed in the frame of DVS, for instance, are static approaches in the sense that they can only exploit the static slack [19], [20], [29]. Nonetheless, there has been a recent interest in dynamic approaches [3], [9], [24], that is, techniques aimed at exploiting the dynamic slack, which is caused by tasks executing less number of clock cycles than their worst case. Solutions by static approaches are computed offline, but have to make pessimistic assumptions, typically in the form

Manuscript received December 2, 2005; revised April 11, 2006. This work was supported by Swedish National Graduate School of Computer Science (CUGS) and by Swedish Foundation for Strategic Research (SSF) under the STRINGENT program.

L. A. Cortés is with the Department of Electrical and Electronics Engineering, Volvo Truck Corporation, Gothenburg, 405 08 Sweden (e-mail: alejandro.cortes@volvo.com).

P. Eles is with the Department of Computer and Information Science, Linköping University, Linköping, 581 83 Sweden (e-mail: petel@ida.liu.se).

Z. Peng is with the Department of Computer Science, Linköping University, Linköping, 581 83 Sweden (e-mail: zebpe@ida.liu.se).

Digital Object Identifier 10.1109/TVLSI.2006.884152

of worst case execution times. Dynamic approaches recompute solutions at runtime in order to exploit the slack that arises from variations in the execution time, but these online computations are typically nontrivial and, consequently, their overhead is high.

In this paper, we focus on realtime systems for which it is possible to tradeoff precision for timeliness as well as energy consumption for performance. We study such systems under the IC model. The goal is to find the voltages at which each task runs and the number of optional cycles, such that the objective function is optimized and the constraints are satisfied. That is, we aim at finding a voltage/optional (V/O)-cycles assignment (actually a set of assignments, as explained later), such that the total reward is maximal while guaranteeing the deadlines and the energy budget. An important contribution of our work is that we exploit the dynamic time and energy slack caused by variations in the actual number of execution cycles. Furthermore, we take into consideration the time and energy overhead incurred during voltage transitions.

In this paper, static V/O assignment refers to finding one assignment of voltages and optional cycles that makes the reward maximal at design-time while guaranteeing the time and energy constraints (this is the problem addressed by [20]). Dynamic V/O assignment refers to finding at runtime, every time a task completes, a new assignment of voltages and optional cycles for those tasks not yet started, but considering the actual execution times by tasks already completed. On the one hand, static V/O assignment causes no online overhead but is rather pessimistic because actual execution times are typically far off from worst case values. On the other hand, dynamic V/O assignment exploits information known only after tasks complete and accordingly computes new assignments; however, the energy and time overhead for online computations is high, even if polynomial-time algorithms can be used. We propose a quasi-static approach that is able to exploit, with low online overhead, the dynamic slack: first, at design-time, a set of V/O assignments are computed and stored (offline phase); second, the selection among the precomputed assignments is left for runtime, based on actual completion times and consumed energy (online phase).

Quasi-static scheduling has been previously studied, mostly in the context of formal synthesis and without considering an explicit notion of time, but only the partial order of events [5], [21], [26]. Recently, in the context of realtime systems, Shih *et al.* have proposed a template-based approach that combines offline and online scheduling for phase array radar systems [22], where templates for schedules are computed offline considering performance constraints, and tasks are scheduled online such that they fit in the templates. The online overhead, though, can be significant when the system workload is high. Quasi-static scheduling for real-time systems with hard and soft tasks was recently discussed [7], but without any energy consideration.

The problem of quasi-static voltage scaling for energy minimization in hard realtime systems was recently addressed [2]. This approach prepares and stores at design-time a number of voltage settings, which are used at runtime for adapting the processor voltage based on actual execution times. Another, somehow similar, approach in which a set of voltage settings is precalculated was discussed in [28]. It considers that the application is given as a task graph composed of subgraphs, some of which might not execute for a certain activation of the system. The selection of a particular voltage setting is, thus, done online based on which subgraphs will be executed at that activation. For each subgraph worst case values are assumed and, consequently, no dynamic slack is exploited. Such approaches, however, target only hard realtime systems and, as opposed to our work, do not consider the reward produced by the soft component of the system (in our case the optional part of tasks).

To the best of our knowledge, the quasi-static approach presented in this paper is the first of its type, that is, it is the first one in which reward, energy, and deadlines are considered together in a quasi-static framework. The chief merit of our approach is its ability to exploit the dynamic slack, caused by tasks completing earlier than in the worst case, at a very low online overhead. This is possible because a set of solutions are prepared and stored at design-time, leaving only for runtime the selection of one of them.

The rest of this paper is structured as follows. Section II introduces notations and definitions used in this paper. In Section III, we motivate the advantages of our approach through an example. The precise formulation of the problem addressed in this paper is presented in Section IV. We propose, in Section V, a method for computing at design-time the set of V/O assignments. Our approach is extensively evaluated in Section VI through numerous synthetic benchmarks and a realistic application. Finally, conclusions are drawn in Section VII.

## II. Preliminaries

### A. Task and Architectural Models

We consider that the functionality of the system is captured by a directed acyclic graph $G = (\mathbf{T}, \mathbf{E})$ where the nodes $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ correspond to the computational tasks and the edges $\mathbf{E}$ indicate the data dependencies between tasks. For the sake of convenience in the notation, we assume that tasks are named according to a particular execution order (as explained later in this subsection) that respects the data dependencies [8]. That is, task $T_{i+1}$ executes immediately after $T_i$, $1 \leq i < n$.

Each task $T_i$ is composed of a mandatory part and an optional part, characterized in terms of the number of CPU cycles $M_i$ and $O_i$, respectively. The actual number of mandatory cycles $M_i$ of a task $T_i$ at a certain activation of the system is unknown beforehand but lies in the interval bounded by the best case number of cycles $M_i^{\mathrm{bc}}$ and the worst case number of cycles $M_i^{\mathrm{wc}}$, that is, $M_i^{\mathrm{bc}} \leq M_i \leq M_i^{\mathrm{wc}}$. The optional part of a task executes immediately after its corresponding mandatory part completes. For each task $T_i$, there is a deadline $d_i$ by which both mandatory and optional parts of $T_i$ must be completed.

For each task $T_i$, there is a reward function $R_i(O_i)$ that takes as an argument the number of optional cycles $O_i$ assigned to $T_i$; we assume that $R_i(0) = 0$. We consider nondecreasing concave[1] reward functions as they capture the particularities of most real

[1]A function $f(x)$ is concave iff $f''(x) \leq 0$, that is, the second derivative is negative.

life applications [20]. Also, as detailed in Section IV, the concavity of reward functions is exploited for obtaining solutions to particular optimization problems in polynomial time. We assume also that there is a value $O_i^{\max}$, for each $T_i$, after which no extra reward is achieved, that is, $R_i(O_i) = R_i^{\max}$ if $O_i \geq O_i^{\max}$. The total reward is the sum of individual reward contributions and is denoted $R = \sum_{T_i \in \mathbf{T}} R_i(O_i)$.

We consider that tasks are nonpreemptable and have equal release time ($r_i = 0, 1 \leq i \leq n$). All tasks are mapped onto a single processor and executed in a fixed order, determined offline, that respects the data dependencies and according to an earliest deadline first (EDF) policy. For nonpreemptable tasks with equal release time and running on a single processor, EDF gives the optimal execution order [6].[2] $T_i$ denotes the $i$-th task in this sequence.

Observe that the nonpreemption assumption is exploited in order to characterize the space from which possible V/O assignments can be selected (as explained in Section V-A) and, thus, obtain a good-quality solution. Note also that our approach deals with voltage levels and optional cycles but not with executing orders. Scheduling and voltage scaling together requires a solution significantly different to the one discussed here and, thus, beyond the scope of this paper.

The target processor supports voltage scaling and we assume that the voltage levels can be varied in a continuous way in the interval $[V^{\min}, V^{\max}]$. If only a discrete set of voltages are supported by the processor, our approach can easily be adapted by using well-known techniques for determining the discrete voltage levels that replace the calculated continuous one [19].

In our quasi-static approach we compute a number of V/O-cycles assignments. The set of precomputed V/O assignments is stored in a dedicated memory in the form of lookup tables, one table $\mathrm{LUT}_i$ for each task $T_i$. The maximum number of V/O assignments that can be stored in memory is a parameter fixed by the designer and is denoted $N^{\max}$.

### B. Energy and Delay Models

The power consumption in CMOS circuits is the sum of dynamic, static (leakage), and short-circuit power. The short-circuit component is negligible. The dynamic power is at the moment the dominating component. However, the leakage power is becoming an important factor in the overall power dissipation. For the sake of simplicity and clarity in the presentation of our ideas, we consider only the dynamic energy consumption. Nonetheless, the leakage energy and adaptive body biasing (ABB) techniques [1] can easily be incorporated into the formulation without changing our general approach. The amount of dynamic energy consumed by task $T_i$ is given by the following expression [16]:

$$E_i = C_i V_i^2 (M_i + O_i) \tag{1}$$

[2]By optimal in this context, we mean the task execution order that, among all feasible orders, admits the V/O assignment which yields the highest total reward. We have demonstrated in [6] that an EDF execution order is the one that least constraints the space of V/O solutions and, henceforth, optimal in the above sense.

where $C_i$ is the effective switched capacitance, $V_i$ is the supply voltage, and $M_i + O_i$ is the total number of cycles executed by the task. The energy overhead caused by switching from $V_i$ to $V_j$ is as follows [16]:

$$\mathcal{E}_{i,j}^{\Delta V} = C_r (V_i - V_j)^2 \tag{2}$$

where $C_r$ is the capacitance of the power rail. We also consider, for the quasi-static solution, the energy overhead $\mathcal{E}_i^{\text{sel}}$ originated from the need to look up and select one of the precomputed V/O assignments. The way we store the precomputed assignments makes the lookup and selection process take $\mathcal{O}(1)$ time. Therefore, $\mathcal{E}_i^{\text{sel}}$ is a constant value. Also, this value is the same for all tasks ($\mathcal{E}_i^{\text{sel}} = \mathcal{E}^{\text{sel}}$, for $1 \leq i \leq n$). For consistency reasons, we keep the index $i$ in the notation of the selection overhead $\mathcal{E}_i^{\text{sel}}$. The energy overhead caused by online operations is denoted $\mathcal{E}_i^{\text{dyn}}$. In a quasi-static solution the online overhead is just the selection overhead ($\mathcal{E}_i^{\text{dyn}} = \mathcal{E}_i^{\text{sel}}$) [8].

The total energy consumed up to the completion of task $T_i$ (including the energy consumed by the tasks themselves as well as the overheads) is denoted $\mathrm{EC}_i = \mathrm{EC}_{i-1} + \mathcal{E}_{i-1,i}^{\Delta V} + \mathcal{E}_i^{\text{dyn}} + E_i$. We consider a given energy budget, denoted $E^{\max}$, that imposes a constraint on the total amount of energy that can be consumed by the system.

The execution time of a task $T_i$ executing $M_i + O_i$ cycles at supply voltage $V_i$ is [16]

$$\tau_i = k \frac{V_i}{(V_i - V_{\text{th}})^\alpha} (M_i + O_i) \tag{3}$$

where $k$ is a constant dependent on the process technology, $\alpha$ is the saturation velocity index (also technology dependent, typically $1.4 \leq \alpha \leq 2$), and $V_{\text{th}}$ is the threshold voltage. The time overhead, when switching from $V_i$ to $V_j$, is given by the following expression [1]:

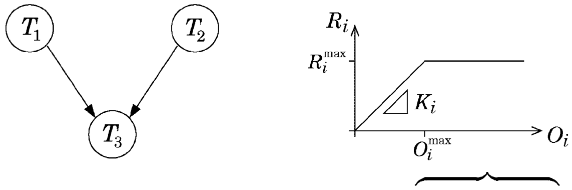
$$\delta_{i,j}^{\Delta V} = p|V_i - V_j| \tag{4}$$

where $p$ is a constant. The time overhead for looking up and selecting one V/O assignment in the quasi-static approach is denoted $\delta_i^{\text{sel}}$ and, as explained above, is constant and is the same value for all tasks.

The starting and completion times of a task $T_i$ are denoted $s_i$ and $t_i$, respectively, with $s_i + \delta_i + \tau_i = t_i$, where $\delta_i$ captures the total time overheads. $\delta_i = \delta_{i-1,i}^{\Delta V} + \delta_i^{\text{dyn}}$, where $\delta_i^{\text{dyn}}$ is the online overhead. Note that in a quasi-static solution this online overhead is just the lookup and selection time, that is, $\delta_i^{\text{dyn}} = \delta_i^{\text{sel}}$.

## III. Motivational Example

Let us consider the motivational example shown in Fig. 1. The nondecreasing reward functions are of the form $R_i(O_i) = K_i O_i$, $O_i \leq O_i^{\max}$. The energy constraint is $E^{\max} = 1$ mJ and the tasks run, according to a schedule fixed offline in conformity to an EDF policy, on a processor that permits continuous voltage scaling in the range 0.6–1.8 V. For clarity reasons, in this example, we assume that transition overheads are zero.




| Task | $M_i^{\mathrm{bc}}$ | $M_i^{\mathrm{wc}}$ | $C_i$ [nF] | $d_i$ [$\mu$s] | $K_i$ | $R_i^{\max}$ |
|---|---|---|---|---|---|---|
| $T_1$ | 20000 | 100000 | 0.7 | 250 | 0.00014 | 7 |
| $T_2$ | 70000 | 160000 | 1.2 | 600 | 0.0002 | 16 |
| $T_3$ | 100000 | 180000 | 0.9 | 1000 | 0.0001 | 6 |

Fig. 1. Motivational example.

TABLE I
OPTIMAL STATIC V/O ASSIGNMENT

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.654 | 35 |
| $T_2$ | 1.450 | 19925 |
| $T_3$ | 1.480 | 11 |

The optimal static V/O assignment for this example is given by Table I. It produces a total reward $R^{\mathrm{st}} = 3.99$. The assignment gives, for each task $T_i$, the voltage $V_i$ at which $T_i$ must run and the number of optional cycles $O_i$ that it must execute in order to obtain the maximum total reward, while guaranteeing that deadlines are met and the energy constraint is satisfied.

The V/O assignment given by Table I is optimal in the static sense. It is the best possible that can be obtained offline without knowing the actual number of cycles executed by each task. However, the actual number of cycles, which are not known in advance, are typically far off from the worst case values used to compute such a static assignment. The implication of such a situation is illustrated by the following case. The first task starts executing at $V_1 = 1.654$ V, as required by the static assignment. Assume that $T_1$ executes $M_1 = 60\,000$ (instead of $M_1^{\mathrm{wc}} = 100\,000$) mandatory cycles and then its assigned $O_1 = 35$ optional cycles. At this point, knowing that $T_1$ has completed at $t_1 = \tau_1 = 111.73$ $\mu$s and that the consumed energy is $\mathrm{EC}_1 = E_1 = 114.97$ $\mu$J, a new V/O assignment can accordingly be computed for the remaining tasks, aiming at obtaining the maximum total reward for the new conditions. We consider, for the moment, the ideal case in which such an online computation takes zero time and energy. Observe that, for computing the new assignments, the worst case for tasks not yet completed has to be assumed as their actual number of executed cycles is not known in advance. The new assignment gives $V_2 = 1.446$ V and $O_2 = 51\,396$. Then $T_2$ runs at $V_2 = 1.446$ V and let us assume that it executes $M_2 = 100\,000$ (instead of $M_2^{\mathrm{wc}} = 160\,000$) mandatory cycles and then its newly assigned $O_2 = 51\,396$ optional cycles. At this point, the completion time is $t_2 = \tau_1 + \tau_2 = 461.35$ $\mu$s and the energy so far consumed is $\mathrm{EC}_2 = E_1 + E_2 = 494.83$ $\mu$J. Again, a new assignment can be computed taking into account the information about completion time and consumed energy. This new assignment gives $V_3 = 1.472$ V and $O_3 = 60\,000$.

For such a situation, in which $M_1 = 60\,000$, $M_2 = 100\,000$, $M_3 = 150\,000$, the V/O assignment computed dynamically (considering $\delta^{\mathrm{dyn}} = 0$ and $\mathcal{E}^{\mathrm{dyn}} = 0$) is summarized in Table II(a). This assignment delivers a total reward $R^{\mathrm{dyn}^{\mathrm{ideal}}} = 16.28$. In reality, however, the online overhead caused by computing new assignments is not negligible. When considering time and energy overheads, using for example $\delta^{\mathrm{dyn}} = 65$ $\mu$s and $\mathcal{E}^{\mathrm{dyn}} = 55$ $\mu$J, the V/O assignment computed dynamically is evidently different, as given by Table II(b). This assignment delivers a total reward $R^{\mathrm{dyn}^{\mathrm{real}}} = 6.26$. The values of $\delta^{\mathrm{dyn}}$ and $\mathcal{E}^{\mathrm{dyn}}$ are, in practice, several orders of magnitude higher than the ones used in this hypothetical example. For instance, for a system with 50 tasks, computing one such V/O assignment using a commercial solver takes a few seconds. Even online heuristics, which produce approximate results, have long execution times [20]. This means that a dynamic V/O scheduler might produce solutions that are actually inferior to the static one (in terms of total reward delivered) or, even worse, a dynamic V/O scheduler might not be able to fulfill the given time and energy constraints, due to the overheads.

TABLE II
DYNAMIC V/O ASSIGNMENTS (FOR $M_1 = 60\,000$, $M_2 = 100\,000$, $M_3 = 150\,000$)

(a)

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.654 | 35 |
| $T_2$ | 1.446 | 51396 |
| $T_3$ | 1.472 | 60000 |

(b)

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.654 | 35 |
| $T_2$ | 1.429 | 1303 |
| $T_3$ | 1.533 | 60000 |

TABLE III
PRECOMPUTED SET OF V/O ASSIGNMENTS

| Task | Condition | $V_i$ [V] | $O_i$ |
|---|---|---|---|
| $T_1$ | — | 1.654 | 35 |
| $T_2$ | if $t_1 \leq 75$ $\mu$s $\wedge$ $EC_1 \leq 77$ $\mu$J | 1.444 | 66924 |
| | else if $t_1 \leq 130$ $\mu$s $\wedge$ $EC_1 \leq 135$ $\mu$J | 1.446 | 43446 |
| | else | 1.450 | 19925 |
| $T_3$ | if $t_2 \leq 400$ $\mu$s $\wedge$ $EC_2 \leq 430$ $\mu$J | 1.380 | 60000 |
| | else if $t_2 \leq 500$ $\mu$s $\wedge$ $EC_2 \leq 550$ $\mu$J | 1.486 | 46473 |
| | else | 1.480 | 11 |

In our quasi-static solution, we compute at design-time a number of V/O assignments, which are selected at runtime by the so-called quasi-static V/O scheduler (at very low overhead) based on the information about completion time and consumed energy after each task completes.

We can define, for instance, a quasi-static set of assignments for the example discussed in this section, as given by Table III. Upon completion of each task, $V_i$ and $O_i$ are selected from the precomputed set of V/O assignments, according to the given condition. The assignments were computed considering the selection overheads $\delta^{\mathrm{sel}} = 0.3$ $\mu$s and $\mathcal{E}^{\mathrm{sel}} = 0.3$ $\mu$J.

For the situation $M_1 = 60\,000$, $M_2 = 100\,000$, $M_3 = 150\,000$ and the set given by Table III, the quasi-static V/O scheduler would do as follows. Task $T_1$ is run at $V_1 = 1.654$ V and is allotted $O_1 = 35$ optional cycles. Since, when completing $T_1$, $t_1 = \tau_1 = 111.73 \leq 130$ $\mu$s and $\mathrm{EC}_1 = E_1 = 114.97 \leq 135$ $\mu$J, $V_2 = 1.446/O_2 = 43\,446$ is selected by the quasi-static V/O scheduler. Task $T_2$ runs under this assignment so that, when it finishes, $t_2 = \tau_1 + \delta_2^{\mathrm{sel}} + \tau_2 = 442.99$ $\mu$s and $\mathrm{EC}_2 = E_1 + \mathcal{E}_2^{\mathrm{sel}} + E_2 = 474.89$ $\mu$J. Then $V_3 = 1.486/O_3 = 46\,473$ is selected and task $T_3$ is executed accordingly. Table IV summarizes the selected assignment. The total reward delivered

TABLE IV
QUASI-STATIC V/O ASSIGNMENT (FOR $M_1 = 60\,000$, $M_2 = 100\,000$, $M_3 = 150\,000$) SELECTED FROM THE PRECOMPUTED SET OF TABLE III

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.654 | 35 |
| $T_2$ | 1.446 | 43446 |
| $T_3$ | 1.486 | 46473 |

by this V/O assignment is $R^{\mathrm{qs}} = 13.34$ (compare to $R^{\mathrm{dyn}^{\mathrm{ideal}}} = 16.28$, $R^{\mathrm{dyn}^{\mathrm{real}}} = 6.26$, and $R^{\mathrm{st}} = 3.99$). It can be noted that the quasi-static solution qsoutperforms the dynamic one $\mathrm{dyn}^{\mathrm{real}}$ because of the large overheads of the latter.

## IV. PROBLEM FORMULATION

We tackle the problem of maximizing the total reward subject to a limited energy budget, in the framework of DVS. In the following, we present the precise formulation of certain related problems and of the particular problem addressed in this paper. Recall that the task execution order is predetermined, with $T_i$ being the $i$th task in this sequence (see Section II-A).

STATIC V/O ASSIGNMENT: Find, for each task $T_i$, $1 \le i \le n$, the voltage $V_i$ and the number of optional cycles $O_i$ that

$$\text{maximize} \quad \sum_{i=1}^{n} R_i(O_i) \tag{5}$$

$$\text{subject to} \quad V^{\min} \le V_i \le V^{\max} \tag{6}$$

$$s_{i+1} = t_i = s_i + \underbrace{p|V_{i-1} - V_i|}_{\delta^{\Delta V}_{i-1,i}} + \underbrace{k\frac{V_i}{(V_i - V_{\mathrm{th}})^{\alpha}}(M_i^{\mathrm{wc}} + O_i)}_{\tau_i} \le d_i \tag{7}$$

$$\sum_{i=1}^{n} \Big(\underbrace{C_r(V_{i-1} - V_i)^2}_{\mathcal{E}^{\Delta V}_{i-1,i}} + \underbrace{C_i V_i^2 (M_i^{\mathrm{wc}} + O_i)}_{E_i}\Big) \le E^{\max}. \tag{8}$$

The previous formulation can be explained as follows. The total reward, as given by (5), is to be maximized. For each task, the voltage $V_i$ must be in the range $[V^{\min}, V^{\max}]$ [(6)]. The completion time $t_i$ is the sum of the start time $s_i$, the voltage-switching time $\delta^{\Delta V}_{i-1,i}$, and the execution time $\tau_i$, and tasks must complete before their deadlines [(7)]. The total energy is the sum of the voltage-switching energies $\mathcal{E}^{\Delta V}_{i-1,i}$ and the energy $E_i$ consumed by each task, and cannot exceed the available energy budget $E^{\max}$ [(8)]. Note that a static assignment must consider the worst case number of mandatory cycles $M_i^{\mathrm{wc}}$ for every task [(7) and (8)].

For tractability reasons, when solving the above problem, we consider $O_i$ as a continuous variable and then we round the result down. By this, without generating the optimal solution, we obtain a solution that is very near to the optimal one because one clock cycle is a very fine-grained unit (tasks execute typically hundreds of thousands of clock cycles). We can rewrite the above problem as "minimize $\sum R'_i(O_i)$," where $R'_i(O_i) = -R_i(O_i)$. It can, thus, be noted that: $R'_i(O_i)$ is convex since $R_i(O_i)$ is a concave function; the constraint functions are also convex.[3] Therefore, we have a convex nonlinear programming (NLP) formulation [27] and, hence, the problem can be solved using polynomial-time methods [18].

*Dynamic V/O Scheduler*: The following is the problem that a dynamic V/O scheduler must solve every time a task $T_c$ completes. It is considered that tasks $T_1, \ldots, T_c$ have already completed (the total energy consumed up to the completion of $T_c$ is $\mathrm{EC}_c$ and the completion time of $T_c$ is $t_c$).

Find $V_i$ and $O_i$, for $c+1 \le i \le n$, that

$$\text{maximize} \quad \sum_{i=c+1}^{n} R_i(O_i) \tag{9}$$

$$\text{subject to} \quad V^{\min} \le V_i \le V^{\max} \tag{10}$$

$$s_{i+1} = t_i = s_i + \delta_i^{\mathrm{dyn}} + \delta^{\Delta V}_{i-1,i} + \tau_i \le d_i \tag{11}$$

$$\sum_{i=c+1}^{n} \left(\mathcal{E}_i^{\mathrm{dyn}} + \mathcal{E}^{\Delta V}_{i-1,i} + E_i\right) \le E^{\max} - \mathrm{EC}_c \tag{12}$$

where $\delta_i^{\mathrm{dyn}}$ and $\mathcal{E}_i^{\mathrm{dyn}}$ are, respectively, the time and energy overhead of computing dynamically $V_i$ and $O_i$ for task $T_i$.

Observe that the problem solved by the dynamic V/O scheduler corresponds to an instance of the static V/O assignment problem (for $c+1 \le i \le n$ and taking into account $t_c$ and $\mathrm{EC}_c$), but considering $\delta_i^{\mathrm{dyn}}$ and $\mathcal{E}_i^{\mathrm{dyn}}$.

The total reward $R^{\mathrm{ideal}}$ delivered by a dynamic V/O scheduler, in the ideal case $\delta_i^{\mathrm{dyn}} = 0$ and $\mathcal{E}_i^{\mathrm{dyn}} = 0$, represents an upper bound on the reward that can practically be achieved without knowing in advance how many mandatory cycles tasks will execute and without accepting risks regarding deadline or energy violations.

Although the dynamic V/O assignment problem can be solved in polynomial-time, the time and energy for solving it are in practice very large and, therefore, unacceptable at runtime for practical applications. In our approach, we prepare offline a number of V/O assignments, one of which is to be selected (at very low online cost) by the *quasi-static V/O scheduler*.

Every time a task $T_c$ completes, the quasi-static V/O scheduler checks the completion time $t_c$ and the total energy $\mathrm{EC}_c$, and looks up an assignment in the table for $T_c$. From the lookup table $\mathrm{LUT}_c$ it obtains the point $(t'_c, \mathrm{EC}'_c)$, which is the closest to $(t_c, \mathrm{EC}_c)$, such that $t_c \le t'_c$ and $\mathrm{EC}_c \le \mathrm{EC}'_c$, and selects $V'/O'$ corresponding to $(t'_c, \mathrm{EC}'_c)$, which are the voltage and number of optional cycles for the next task $T_{c+1}$. Our aim is to obtain a reward, as delivered by the quasi-static V/O scheduler, that is maximal. This problem we discuss in the rest of the paper as follows.

[3]Observe that the function *abs* cannot be used directly in mathematical programming because it is not differentiable in 0. However, there exist established techniques for obtaining equivalent formulations that can be used in NLP problems [1].

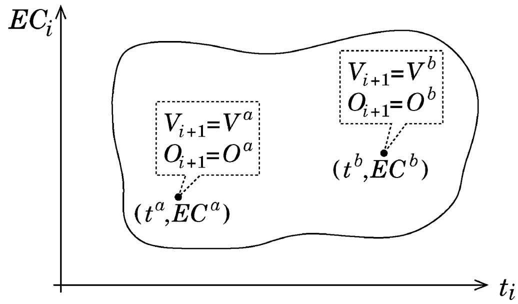



Fig. 2. Time-energy space.

*Set of V/O Assignments*: Find a set of $N$ assignments such that: $N \leq N^{\max}$; the V/O assignment selected by the quasi-static V/O scheduler guarantees the deadlines $(s_i + \delta_i^{\text{sel}} + \delta_{i-1,i}^{\Delta V} + \tau_i = t_i \leq d_i)$ and the energy constraint $(\sum_{i=1}^{n} \mathcal{E}_i^{\text{sel}} + \mathcal{E}_{i-1,i}^{\Delta V} + E_i \leq E^{\max})$, and yields a total reward $R^{\text{qs}}$ that is maximal.

As will be discussed in Section V, for a task $T_i$, potentially there exist infinitely many possible values for $t_i$ and $\text{EC}_i$. Therefore, in order to approach the theoretical limit $R^{\text{ideal}}$, it would be needed to compute an infinite number of V/O assignments, one for each $(t_i, \text{EC}_i)$. The problem is, thus, how to select at most $N^{\max}$ points in this infinite space such that the respective V/O assignments produce a reward as close as possible to $R^{\text{ideal}}$.

## V. Computing the Set of V/O Assignments

For each task $T_i$, there exists a time-energy space of possible values of completion time $t_i$ and energy $\text{EC}_i$ consumed up to the completion of $T_i$ (see Fig. 2). Every point in this space defines a V/O assignment for the next task $T_{i+1}$, that is, if $T_i$ completed at $t^a$ and the energy consumed was $\text{EC}^a$, the assignment for the next task would be $V_{i+1} = V^a / O_{i+1} = O^a$. The values $V^a$ and $O^a$ are those that an ideal dynamic V/O scheduler would give for the case $t_i = t^a, \text{EC}_i = \text{EC}^a$ (recall that we aim at matching the reward $R^{\text{ideal}}$). Observe that different points $(t_i, \text{EC}_i)$ define different V/O assignments. Note also that for a given value $t_i$ there might be different valid values of $\text{EC}_i$, and this is due to the fact that different previous V/O assignments can lead to the same $t_i$ but still different $\text{EC}_i$.

We need first to determine the boundaries of the time-energy space for each task $T_i$, in order to select $N_i$ points in this space and accordingly compute the set of $N_i$ assignments. $N_i$ is the number of assignments to be stored in the lookup table $\text{LUT}_i$, after distributing the maximum number $N^{\max}$ of assignments among tasks. A straightforward way to determine these boundaries is to compute the earliest and latest completion times as well as the minimum and maximum consumed energy for task $T_i$, based on the values $V^{\min}$, $V^{\max}$, $M_j^{\text{bc}}$, $M_j^{\text{wc}}$, and $O_j^{\max}$, $1 \leq j \leq i$. The earliest completion time $t_i^{\min}$ occurs when each of the previous tasks $T_j$ (inclusive $T_i$) execute their minimum number of cycles $M_j^{\text{bc}}$ and zero optional cycles at maximum voltage $V^{\max}$, while $t_i^{\max}$ occurs when each task $T_j$ executes $M_j^{\text{wc}} + O_j^{\max}$ cycles at $V^{\min}$. Similarly, $\text{EC}_i^{\min}$ happens when each task $T_j$ executes $M_j^{\text{bc}}$ cycles at $V^{\min}$, while $\text{EC}_i^{\max}$ happens when each task $T_j$ executes $M_j^{\text{wc}} + O_j^{\max}$ cycles at $V^{\max}$. The intervals $[t_i^{\min}, t_i^{\max}]$ and $[\text{EC}_i^{\min}, \text{EC}_i^{\max}]$ bound the time-energy space as shown

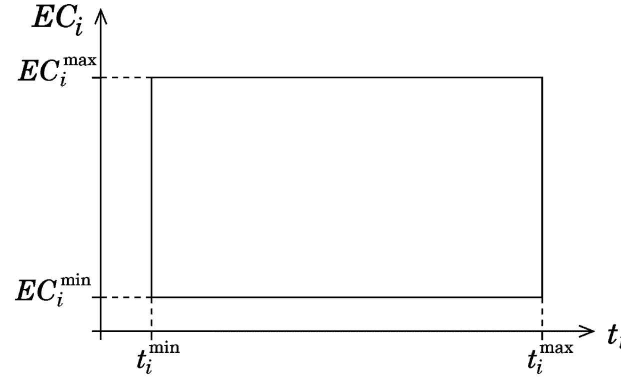



Fig. 3. Pessimistic boundaries of the time-energy space.

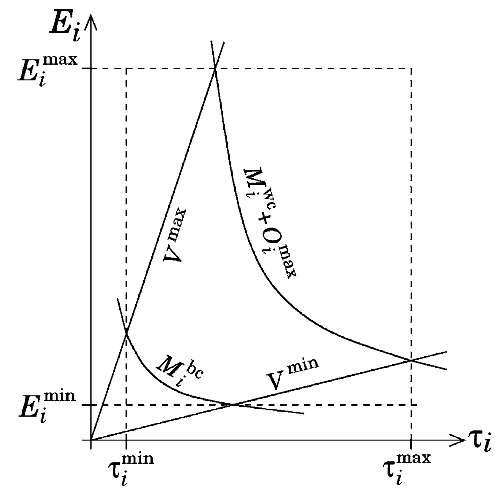



Fig. 4. $\tau_i$-$E_i$ space for task $T_i$.

in Fig. 3. However, there are points in this space that cannot happen. For instance, $(t_i^{\min}, \text{EC}_i^{\min})$ is not feasible because $t_i^{\min}$ requires all tasks running at $V^{\max}$ whereas $\text{EC}_i^{\min}$ requires all tasks running at $V^{\min}$.

### A. Characterization of the Time-Energy Space

In this section, we make a characterization of the time-energy space in order to determine how the points in this space (for which the V/O assignments are to be computed offline) should be selected so that good quality results are achieved. We elaborate on the different steps along this characterization.

We now take a closer look at the relation between the energy $E_i$ consumed by a task and its execution time $\tau_i$ as given, respectively, by (1) and (3). In this section, we consider that the execution time is inversely proportional to the supply voltage $(V_{\text{th}} = 0, \alpha = 2)$, an assumption commonly made in the literature [19]. Observe, however, that we make such an assumption only in order to make the illustration of our point simpler, yet the drawn conclusions are valid, in general, and do not rely on this assumption.

After some simple algebraic manipulations on (1) and (3), we get the following expression:

$$E_i = \frac{C_i V_i^3}{k} \tau_i \tag{13}$$

which, in the $\tau_i$-$E_i$ space, gives a family of straight lines, each for a particular $V_i$. Thus, $E_i = C_i (V^{\min})^3 \tau_i / k$ and $E_i =$

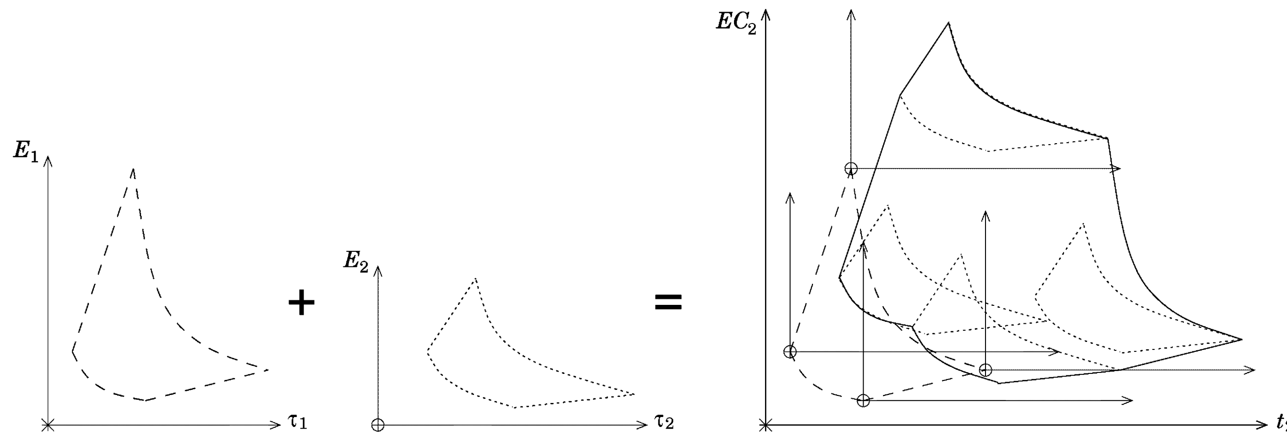



Fig. 5. Illustration of the "sum" of spaces $\tau_1$-$E_1$ and $\tau_2$-$E_2$.

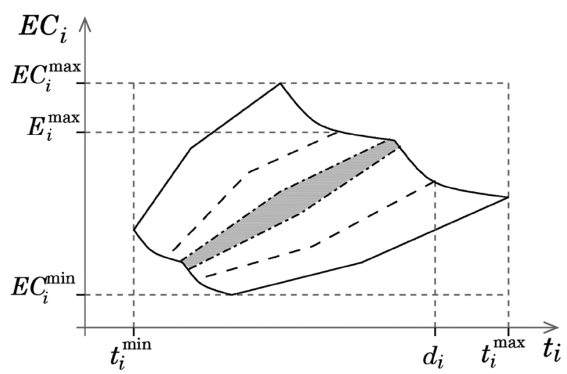



Fig. 6. Realistic boundaries of the time-energy space.

$C_i(V^{\max})^3\tau_i/k$ define two boundaries in the $\tau_i$-$E_i$ space. We can also write the following equation:

$$E_i = C_i k^2 (M_i + O_i)^3 \frac{1}{\tau_i^2} \tag{14}$$

which gives a family of curves, each for a particular $M_i + O_i$. Thus, $E_i = C_i k^2 (M_i^{\text{bc}})^3/\tau_i^2$ and $E_i = C_i k^2 (M_i^{\text{wc}} + O_i^{\max})^3/\tau_i^2$ define other two boundaries, as shown in Fig. 4. Note that Fig. 4 represents the energy consumed by one task (energy $E_i$ if $T_i$ executes for $\tau_i$ time), as opposed to Fig. 3 that represents the energy by all tasks up to $T_i$ (total energy $\text{EC}_i$ consumed up to the moment $t_i$ when task $T_i$ finishes).

In order to obtain a realistic, less pessimistic, view of the diagram in Fig. 3, we must "sum" the spaces $\tau_j$-$E_j$ previously introduced. The result of this summation, as illustrated in Fig. 5, gives the time-energy space $t_i$-$\text{EC}_i$ we are interested in. In Fig. 5, the $t_2$-$\text{EC}_2$ space is obtained by sliding the $\tau_2$-$E_2$ space with its coordinate origin along the boundaries of $\tau_1$-$E_1$. The "south-east" (SE) and "north-west" (NW) boundaries of the $t_i$-$\text{EC}_i$ space are piecewise linear because the SE and NW boundaries of the individual spaces $\tau_j$-$E_j$, $1 \leq j \leq i$, are straight lines (see Fig. 4). Similarly, the NE and SW boundaries of the $t_i$-$\text{EC}_i$ space are piecewise parabolic because the NE and SW of the individual spaces $\tau_j$-$E_j$ are parabolic. The shape of the $t_i$-$\text{EC}_i$ space, obtained as a result of such a summation, is depicted by the solid lines in Fig. 6.

In order to further refine the $t_i$-$\text{EC}_i$ space, one has to consider, in addition, deadlines $d_i$ as well as energy constraints $E_i^{\max}$. Note that, for each task, the deadline $d_i$ is explicitly given as part of the system model, whereas $E_i^{\max}$ is an implicit constraint induced by the overall energy constraint $E^{\max}$. The energy constraint $E_i^{\max}$ imposed upon the completion of task $T_i$ comes from the fact that future tasks will consume at least a certain amount of energy $F_{i+1\to n}$ so that $E_i^{\max} = E^{\max} - F_{i+1\to n}$. The deadline $d_i$ and the induced energy constraint $E_i^{\max}$ further restrict the time-energy space, as depicted by the dashed lines in Fig. 6.

The time-energy space can be narrowed down even further if we take into consideration that we are only interested in points as calculated by the ideal dynamic V/O scheduler. This is explained in the following. Let us consider two different activations of the system. In the first one, after finishing task $T_{i-1}$ at $t'_{i-1}$ with a total consumed energy $\text{EC}'_{i-1}$, task $T_i$ runs under a certain assignment $V'_i/O'_i$. In the second activation, after $T_{i-1}$ completes at $t''_{i-1}$ with total energy $\text{EC}''_{i-1}$, $T_i$ runs under the assignment $V''_i/O''_i$. Since the points $(t'_{i-1}, \text{EC}'_{i-1})$ and $(t''_{i-1}, \text{EC}''_{i-1})$ are, in general, different, the assignments $V'_i/O'_i$ and $V''_i/O''_i$ are also different. The assignment $V'_i/O'_i$ defines in the $t_i$-$\text{EC}_i$ space a segment of straight line $L'_i$ that has slope $C_i(V'_i)^3/k$, with one end point corresponding to the execution of $M_i^{\text{bc}} + O'_i$ cycles at $V'_i$ and the other end corresponding to the execution of $M_i^{\text{wc}} + O'_i$ cycles at $V'_i$ [see (13)]. $V''_i/O''_i$ defines analogously a different segment of straight line $L''_i$. The lines $L'_i$ and $L''_i$ defined, respectively, by the assignments $V'_i/O'_i$ and $V''_i/O''_i$ are depicted in Fig. 7. It must be observed that solutions to the dynamic V/O assignment problem, though, tend toward letting tasks consume as much as possible of the available energy and finish as late as possible without risking energy or deadline violations: there is no gain by consuming less energy or finishing earlier than needed, as the goal is to maximize the reward. Both solutions $V'_i/O'_i$ and $V''_i/O''_i$ (that is, the straight lines $L'_i$ and $L''_i$ in Fig. 7) will, thus, converge in the $t_i$-$\text{EC}_i$ space when $M'_i = M''_i = M_i^{\text{wc}}$ (which is the value of mandatory cycles that has to be assumed when computing the V/O values to be assigned to $T_i$ after $T_{i-1}$ has terminated). Therefore, if $T_i$ under the assignment $V'_i/O'_i$ completes at the same time as under the second assignment $V''_i/O''_i$ $(t'_i = t''_i)$, the respective energy values $\text{EC}'_i$ and $\text{EC}''_i$

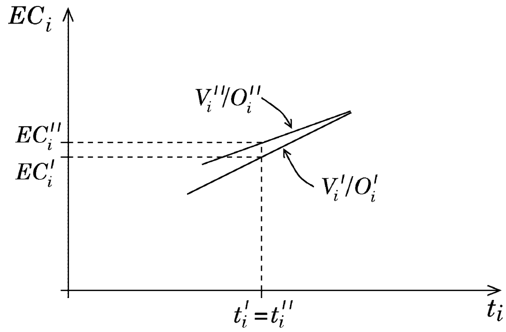



Fig. 7. $V_i'/O_i'$ and $V_i''/O_i''$ converge.

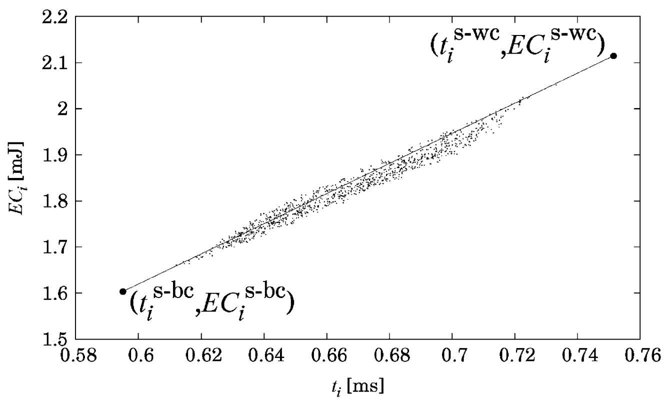



Fig. 8. Actual points in the time-energy space.

will actually be very close as shown in Fig. 7. This means that in practice the $t_i$-$\mathrm{EC}_i$ space is a narrow area, as depicted by the dash-dot lines and the gray area enclosed by them in Fig. 6.

We have conducted a number of experiments in order to determine how narrow the area of points in the time-energy space actually is. For each task $T_i$, we consider a segment of straight line, called in the sequel the diagonal $D_i$, defined by the points $(t_i^{s-\mathrm{bc}}, \mathrm{EC}_i^{s-\mathrm{bc}})$ and $(t_i^{s-\mathrm{wc}}, \mathrm{EC}_i^{s-\mathrm{wc}})$. The point $(t_i^{s-\mathrm{bc}}, \mathrm{EC}_i^{s-\mathrm{bc}})$ corresponds to the solution given by the ideal dynamic V/O scheduler in the particular case when every task $T_j$, $1 \leq j \leq i$, executes its best case number of mandatory cycles $M_j^{\mathrm{bc}}$. Analogously, $(t_i^{s-\mathrm{wc}}, \mathrm{EC}_i^{s-\mathrm{wc}})$ corresponds to the solution in the particular case when every task $T_j$ executes its worst case number of mandatory cycles $M_j^{\mathrm{wc}}$. We have generated 50 synthetic examples, consisting of between 10 and 100 tasks, and simulated for each of them the ideal dynamic V/O scheduler for 1000 cases, each case $S$ being a combination of executed mandatory cycles $M_1^S, M_2^S, \ldots, M_n^S$. Note that we have used various distributions for the number of mandatory cycles. For each task $T_i$ of the different benchmarks and for each set $S$ of mandatory cycles, we obtained the actual point $(t_i^S, \mathrm{EC}_i^S)$ in the $t_i$-$\mathrm{EC}_i$ space, as given by the ideal dynamic V/O scheduler. Each point $(t_i^S, \mathrm{EC}_i^S)$ was compared with the point $(t_i^S, \mathrm{EC}_i^{D_i})$ (a point with the same abscissa $t_i^S$, but on the diagonal $D_i$ so that its ordinate is $\mathrm{EC}_i^{D_i}$) and the relative deviation $e = |\mathrm{EC}_i^S - \mathrm{EC}_i^{D_i}|/\mathrm{EC}_i^S$ was computed. We found through our experiments average deviations of around 1% and a maximum deviation of 4.5%. These results show that the $t_i$-$\mathrm{EC}_i$ space is indeed a narrow area. For example, Fig. 8 shows the actual points $(t_i^S, \mathrm{EC}_i^S)$, corresponding to the simulation of the 1000 sets $S$ of executed mandatory cycles, in the time-energy space of a particular task $T_i$ as well as the diagonal $D_i$. It can also be mentioned that Fig. 8 corresponds to an experiment where the number of mandatory cycles $M_i$ were considered to be uniformly distributed in its interval $[M_i^{\mathrm{bc}}, M_i^{\mathrm{wc}}]$.

| Condition | $V_{i+1}$ | $O_{i+1}$ |
|---|---|---|
| if $t_i \leq t^a \wedge EC_i \leq EC^a$ | $V^a$ | $O^a$ |
| else if $t_i \leq t^b \wedge EC_i \leq EC^b$ | $V^b$ | $O^b$ |
| else if $t_i \leq t^c \wedge EC_i \leq EC^c$ | $V^c$ | $O^c$ |
| else | $V^d$ | $O^d$ |

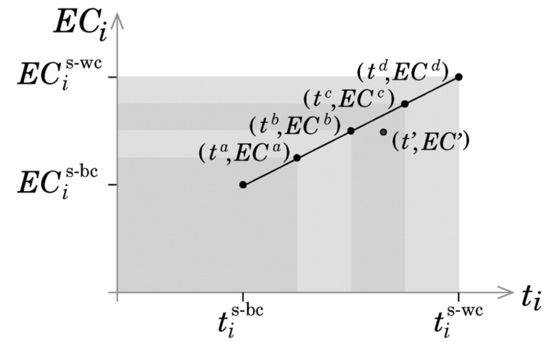



Fig. 9. Regions.

### B. Selection of Points and Computation of Assignments

From the discussion in Section V-A, we can draw the conclusion that the points in the $t_i$-$\mathrm{EC}_i$ space are concentrated in a relatively narrow area along the diagonal $D_i$. This observation is crucial for selecting the points for which we compute, at design-time, the V/O assignments.

We take $N_i$ points $(t_i^j, \mathrm{EC}_i^j)$ along the diagonal $D_i$ in the $t_i$-$\mathrm{EC}_i$ space of task $T_i$, and then we compute and store the respective assignments $V_{i+1}^j/O_{i+1}^j$ that maximize the total reward when $T_i$ completes at $t_i^j$ and the total energy is $\mathrm{EC}_i^j$. It should be noted that for the computation of the assignment $V_{i+1}^j/O_{i+1}^j$, the time and energy overheads $\delta_{i+1}^{\mathrm{sel}}$ and $\mathcal{E}_{i+1}^{\mathrm{sel}}$ (needed for selecting assignments at run time) are taken into account.

Each one of the points, together with its corresponding V/O assignment, covers a region as indicated in Fig. 9. The quasi-static V/O scheduler selects one of the stored assignments based on the actual completion time and consumed energy. Referring to Fig. 9, for example, if task $T_i$ completes at $t'$ and the consumed energy is $\mathrm{EC}'$, the quasi-static V/O scheduler will select the precomputed V/O assignment corresponding to $(t^c, \mathrm{EC}^c)$.

The pseudocode of the procedure we use for computing, offline, the set of V/O assignments is given by Algorithm 1. First, the maximum number $N^{\max}$ of assignments that are to be stored is distributed among tasks (line 1). A straightforward approach is to distribute them uniformly among the different tasks, so that each lookup table contains the same number of assignments. However, as shown by the experimental evaluation of Section VI, it is more efficient to distribute the assignments according to the size of the time-energy space of tasks (we use the length of the diagonal $D$ as a measure of this size), in such a way that the lookup tables of tasks with larger spaces get more points.

After distributing $N^{\max}$ among tasks, the solutions $\mathrm{V/O}^{s-\mathrm{bc}}$ and $\mathrm{V/O}^{s-\mathrm{wc}}$ are computed (lines 2-3). $\mathrm{V/O}^{s-\mathrm{bc}}$ ($\mathrm{V/O}^{s-\mathrm{wc}}$) is a structure that contains the pairs $V_i^{s-\mathrm{bc}}/O_i^{s-\mathrm{bc}}$ ($V_i^{s-\mathrm{wc}}/O_i^{s-\mathrm{wc}}$), for each task $T_i$, $1 \leq i \leq n$. $V_i^{s-\mathrm{bc}}/O_i^{s-\mathrm{bc}}$

and $V_i^{s-\mathrm{wc}}/O_i^{s-\mathrm{wc}}$ are the values that the dynamic V/O scheduler would compute when every task executes its best and worst case number of cycles, respectively. In order to obtain the values $V_i^{s-\mathrm{bc}}/O_i^{s-\mathrm{bc}}$ and $V_i^{s-\mathrm{wc}}/O_i^{s-\mathrm{wc}}$, $1 \le i \le n$ (and, thus, the structures $\mathrm{V/O}^{s-\mathrm{bc}}$ and $\mathrm{V/O}^{s-\mathrm{wc}}$), we solve the problem, as formulated by (9)–(12), $n-1$ times (corresponding to each task completion) first, considering the best case of mandatory cycles for all tasks, and second, considering the worst case of mandatory cycles for all tasks.

Since the assignment $V_1/O_1$ is invariably the same (line 4), this is the only one stored for the first task (line 5).

For every task $T_i$, $1 \le i \le n-1$, we compute: 1) $t_i^{s-\mathrm{bc}}$ ($t_i^{s-\mathrm{wc}}$) as the sum of execution times $\tau_k^{s-\mathrm{bc}}$ ($\tau_k^{s-\mathrm{wc}}$)—given by (3) with $V_k^{s-\mathrm{bc}}$, $M_k^{\mathrm{bc}}$, and $O_k^{s-\mathrm{bc}}$ ($V_k^{s-\mathrm{wc}}$, $M_k^{\mathrm{wc}}$, and $O_k^{s-\mathrm{wc}}$)—and time overheads $\delta_k$ (line 7); 2) $\mathrm{EC}_i^{s-\mathrm{bc}}$ ($\mathrm{EC}_i^{s-\mathrm{wc}}$) as the sum of energies $E_k^{s-\mathrm{bc}}$ ($E_k^{s-\mathrm{wc}}$)—given by (1) with $V_k^{s-\mathrm{bc}}$, $M_k^{\mathrm{bc}}$, and $O_k^{s-\mathrm{bc}}$ ($V_k^{s-\mathrm{wc}}$, $M_k^{\mathrm{wc}}$, and $O_k^{s-\mathrm{wc}}$)—and energy overheads $\mathcal{E}_k$ (line 8).

Having the endpoints $(t_i^{s-\mathrm{bc}}, \mathrm{EC}_i^{s-\mathrm{bc}})$ and $(t_i^{s-\mathrm{wc}}, \mathrm{EC}_i^{s-\mathrm{wc}})$ that define the diagonal $D_i$ (as shown in Fig. 9), for every task $T_i$, we take $N_i$ equally-spaced points $(t_i^j, \mathrm{EC}_i^j)$ along $D_i$ and, for each such point, we compute the respective assignment $V_{i+1}^j/O_{i+1}^j$ (we solve the dynamic V/O assignment problem as formulated by (9)–(12), assuming that the total energy consumed up to the completion of $T_i$ is $\mathrm{EC}_i^j$ and the completion time of $T_i$ is $t_i^j$) and store it accordingly in the $j$th position in the particular lookup table $\mathrm{LUT}_i$ (lines 10–12).

**Algorithm 1: OffLinePhase**

**input:** The maximum number $N^{\max}$ of assignments
**output:** The set of V/O assignments
1: distribute $N^{\max}$ among tasks ($T_i$ gets $N_i$ points)
2: $\mathrm{V/O}^{s-\mathrm{bc}}$ := sol. by dyn. scheduler when $M_k = M_k^{\mathrm{bc}}$, $1 \le k \le n$
3: $\mathrm{V/O}^{s-\mathrm{wc}}$ := sol. by dyn. scheduler when $M_k = M_k^{\mathrm{wc}}$, $1 \le k \le n$
4: $V_1 := V_1^{s-\mathrm{bc}} = V_1^{s-\mathrm{wc}}$; $O_1 := O_1^{s-\mathrm{bc}} = O_1^{s-\mathrm{wc}}$
5: store $V_1/O_1$ in $\mathrm{LUT}_1[1]$
6: **for** $i \leftarrow 1, 2, \ldots, n-1$ **do**
7: $\quad t_i^{s-\mathrm{bc}} := \sum_{k=1}^{i}(\tau_k^{s-\mathrm{bc}} + \delta_k)$; $t_i^{s-\mathrm{wc}} := \sum_{k=1}^{i}(\tau_k^{s-\mathrm{wc}} + \delta_k)$
8: $\quad \mathrm{EC}_i^{s-\mathrm{bc}} := \sum_{k=1}^{i}(E_k^{s-\mathrm{bc}} + \mathcal{E}_k)$; $\mathrm{EC}_i^{s-\mathrm{wc}} := \sum_{k=1}^{i}(E_k^{s-\mathrm{wc}} + \mathcal{E}_k)$
9: $\quad$ **for** $j \leftarrow 1, 2, \ldots, N_i$
10: $\quad\quad t_i^j := [(N_i - j)t_i^{s-\mathrm{bc}} + j\, t_i^{s-\mathrm{wc}}]/N_i$
11: $\quad\quad \mathrm{EC}_i^j := [(N_i - j)\mathrm{EC}_i^{s-\mathrm{bc}} + j\mathrm{EC}_i^{s-\mathrm{wc}}]/N_i$
12: $\quad\quad$ compute $V_{i+1}^j/O_{i+1}^j$ for $(t_i^j, \mathrm{EC}_i^j)$ and store it in $\mathrm{LUT}_i[j]$
13: $\quad$ **end for**
14: **end for**

The set of V/O assignments, prepared offline, is used online by the quasi-static V/O scheduler as outlined by Algorithm 2. Note that the energy $\mathrm{EC}_i$ consumed up to the completion of task $T_i$ can be calculated based on the energy $\mathrm{EC}_{i-1}$ consumed up to the previous task and the actual execution time $\tau_i$ as given by line 1. Upon completing task $T_i$, the lookup table $\mathrm{LUT}_i$ is consulted. If the point $(t, \mathrm{EC})$ lies above the diagonal $D_i$ (line 2) the index $j$ of the table entry is simply calculated as in line 3, else as in line 5. Computing directly the index $j$, instead of searching through the table $\mathrm{LUT}_i$, is possible because the points $(t_i^j, \mathrm{EC}_i^j)$ stored in $\mathrm{LUT}_i$ are equally-spaced. Finally, the V/O assignment stored in $\mathrm{LUT}_i[j]$ is retrieved (line 7). Observe that Algorithm 2 has a time complexity $\mathcal{O}(1)$ and, therefore, the online operation performed by the quasi-static V/O scheduler takes constant time and energy. Also, this lookup and selection process is several orders of magnitude cheaper than the online computation by the dynamic V/O scheduler.

**Algorithm 2: OnLinePhase**

**input:** Actual completion time $t = t_i$ of $T_i$, and lookup table $\mathrm{LUT}_i$ (contains $N_i$ assignments and the diagonal $D_i$—defined as $\mathrm{EC}_i = A_i t_i + B_i$)
**output:** The assignment $V_{i+1}/O_{i+1}$ for the next task $T_{i+1}$
1: $EC = \mathrm{EC}_i = \mathrm{EC}_{i-1} + \mathcal{E}_{i-1,i}^{\Delta V} + \mathcal{E}^{\mathrm{sel}} + C_i V_i \frac{(V_i - V_{th})^{\alpha}}{k} \tau_i$
2: **if** $EC > A_i t + B_i$ **then**
3: $\quad j := \lceil N_i(\mathrm{EC} - \mathrm{EC}_i^{s-\mathrm{bc}})/(\mathrm{EC}_i^{s-\mathrm{wc}} - \mathrm{EC}_i^{s-\mathrm{bc}}) \rceil$
4: **else**
5: $\quad j := \lceil N_i(t - t_i^{s-\mathrm{bc}})/(t_i^{s-\mathrm{wc}} - t_i^{s-\mathrm{bc}}) \rceil$
6: **end if**
7: return V/O assignment stored in $\mathrm{LUT}_i[j]$

## VI. Experimental Evaluation

In order to evaluate the presented approach, we performed a large number of experiments using numerous synthetic benchmarks. We generated task graphs containing between 10 and 100 tasks. Each point in the plots of the experimental results (Figs. 10–12) corresponds to 50 automatically-generated task graphs, resulting overall in more than 4000 performed evaluations. The technology-dependent parameters were adopted from [16] and correspond to a processor in a 0.18-$\mu$m CMOS fabrication process. The reward functions we used along the experiments are of the form $R_i(O_i) = \alpha_i O_i + \beta_i \sqrt{O_i} + \gamma_i \sqrt{[3]} O_i$, with coefficients $\alpha_i$, $\beta_i$, and $\gamma_i$ randomly chosen. The experiments in this section were performed using uniform probability distributions for the number of mandatory cycles.

The first set of experiments was performed with the goal of investigating the reward gain achieved by our quasi-static approach compared to the optimal static solution (the approach proposed in [20]). In these experiments we consider that the time and energy overheads needed for selecting the assignments by the quasi-static V/O scheduler are $\delta^{\mathrm{sel}} = 450$ ns and $\mathcal{E}^{\mathrm{sel}} = 400$ nJ. These are realistic values as selecting a precomputed assignment takes only tens of cycles and the access time and energy consumption (per access) of, for example, a low-power Static RAM are around 70 ns and 20 nJ, respectively, [17].[4] Fig. 10(a) shows the reward (normalized with respect to the

[4]Besides, in order to get a realistic estimation of the number of clock cycles needed for executing Algorithm 2, we used the MPARM simulation environment [4]: we found out that it requires about 200 clock cycles in a processor with floating-point unit, which in the case of the Crusoe 5600 processor at 600 MHz means less than 350 ns.

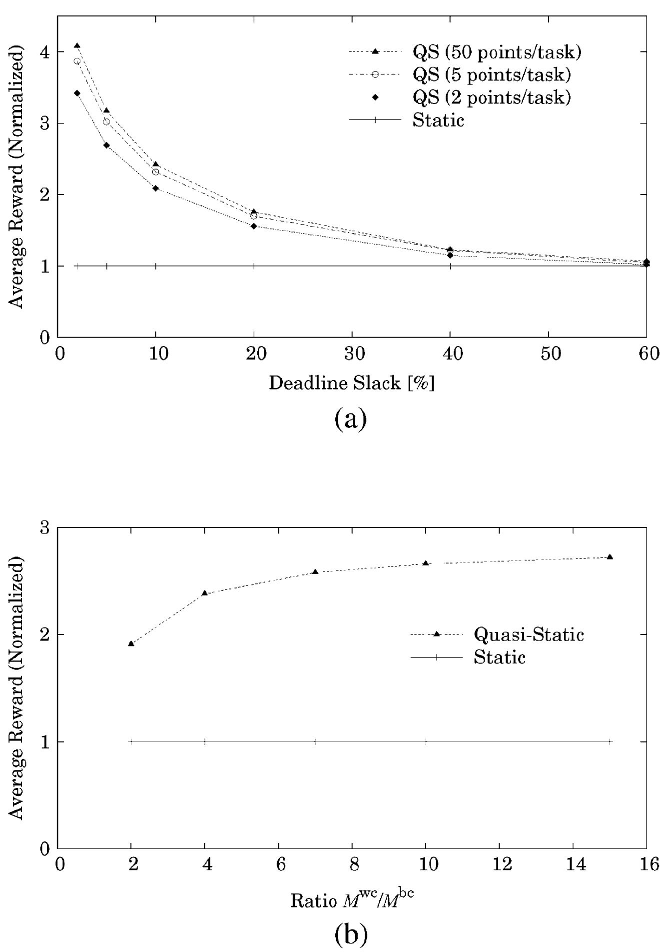



Fig. 10. Comparison of the quasi-static and static solutions. (a) Influence of the deadline slack. (b) Influence of the ratio $M^{\rm wc}/M^{\rm bc}$.

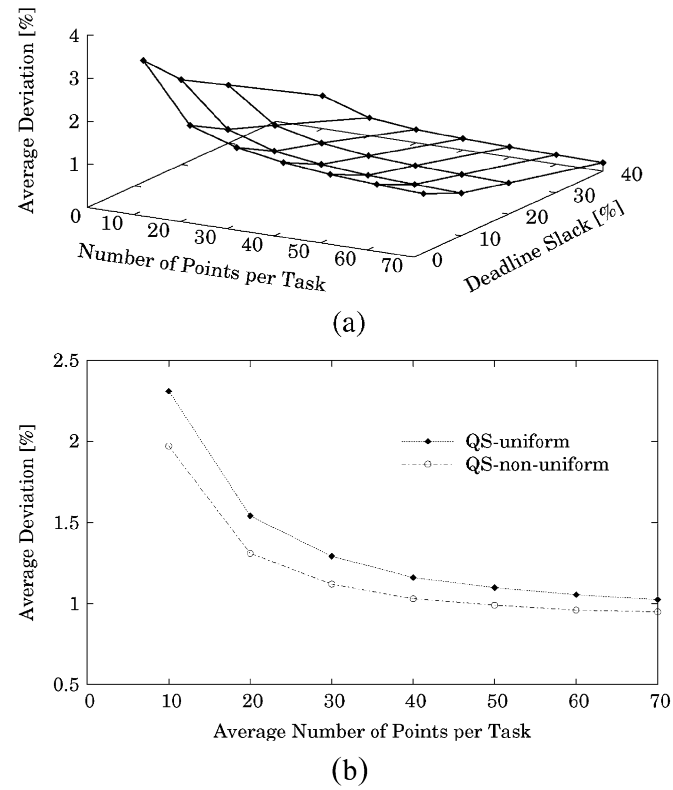



Fig. 11. Comparison of the quasi-static and ideal dynamic solutions. (a) Influence of the deadline slack and number of points. (b) Influence of the distribution of points among lookup tables.

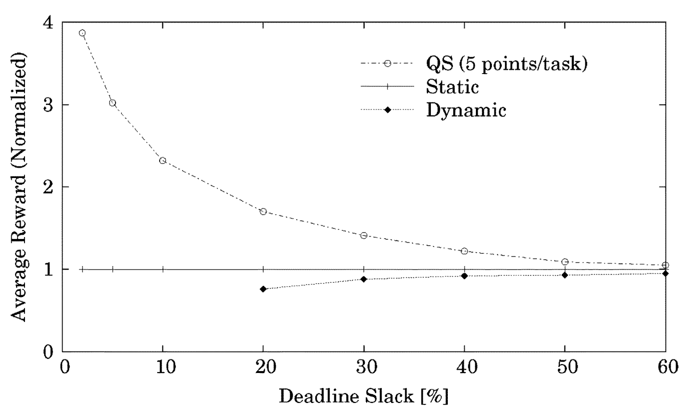



Fig. 12. Comparison considering realistic overheads.

reward given by the static solution) as a function of the deadline slack (the relative difference between the deadline and the completion time when worst case number of mandatory cycles are executed at the maximum voltage that guarantees the energy constraint). Three cases for the quasi-static approach (2, 5, and 50 points per task) are considered in this figure. Very significant gains in terms of total reward, up to four times, can be obtained with the quasi-static solution, even with few points per task. The highest reward gains are achieved when the system has very tight deadlines (small deadline slack). This is so because, when large amounts of slack are available, the static solution can accommodate most of the optional cycles (recall there is a value $O_i^{\max}$ after which no extra reward is achieved) and, therefore, the difference in reward between the static and quasi-static solutions is not big in these cases.

The influence of the ratio between the worst case number of cycles $M^{\rm wc}$ and the best case number of cycles $M^{\rm bc}$ has also been studied and the results are presented in Fig. 10(b). In this case, we have considered systems with a deadline slack of 10% and 20 points per task in the quasi-static solution. The larger the ratio $M^{\rm wc}/M^{\rm bc}$ is, the more the actual number of execution cycles deviate from the worst case value $M^{\rm wc}$ (which is the value that has to be considered by a static solution). Thus, the dynamic slack becomes larger and, therefore, there are more chances to exploit such a slack and, consequently, improve the reward.

The second set of experiments was aimed at evaluating the quality of our quasi-static approach with respect to the theoretical limit that could be achieved without knowing in advance the exact number of execution cycles (the reward delivered by the ideal dynamic V/O scheduler, as discussed in Section IV, in which the dynamic V/O assignment problem formulated by (9)–(12) is solved assuming zero overheads $\delta_i^{\rm dyn}$ and $\mathcal{E}_i^{\rm dyn}$). For the sake of comparison fairness, since the reward produced by an ideal dynamic V/O scheduler with no overheads is taken as reference point, in this set of experiments, we have also considered zero time and energy overheads $\delta^{\rm sel}$ and $\mathcal{E}^{\rm sel}$ for the proposed quasi-static approach (as opposed to the previous experiments).

Fig. 11(a) shows the deviation $\mathrm{dev} = (R^{\mathrm{ideal}} - R^{\mathrm{qs}})/R^{\mathrm{ideal}}$ as a function of the number of precomputed assignments (points per task) and for various degrees of deadline tightness. The ordinate in Fig. 11(a), as well as in Fig. 11(b), shows the average deviation of the reward given by a quasi-static approach in relation to the reward produced by a dynamic V/O scheduler in the ideal case of zero time and energy overheads. More points per task produce higher reward, closer to the theoretical limit (smaller deviation). Nonetheless, with relatively few points per task, we can still get very close to the theoretical limit, for instance, in systems with deadline slack of 20% and for 30 points per task the average deviation is around 1.3%. As mentioned previously, when the deadline slack is large even a static solution (which corresponds to a quasi-static solution with just one point per task) can accommodate most of the optional cycles. Hence, the deviation gets smaller as the deadline slack increases, as shown in Fig. 11(a).

In the previous experiments, it has been considered that, for a given system, the lookup tables have the same size, that is, contain the same number of assignments. When the number $N^{\max}$ of assignments is distributed among tasks according to the size of their time-energy spaces (more assignments in the lookup tables of tasks that have larger spaces), better results are obtained, as shown in Fig. 11(b). This figure plots the case of equal-size lookup tables (QS-uniform) and the case of assignments distributed nonuniformly among tables (QS-nonuniform), as described above, for systems with a deadline slack of 20%. The abscissa is the average number of points per task.

In a third set of experiments, we took into account the online overheads of the dynamic V/O scheduler (as well as the quasi-static one) and compared the static, quasi-static, and dynamic approaches in the same graph. Fig. 12 shows the total reward normalized with respect to the one produced by the static solution. It shows that, in a realistic setting, the dynamic approach performs poorly, even worse than the static one. More importantly, for systems with tight deadlines (small deadline slack), the dynamic approach cannot guarantee the time and energy constraints because of its large overheads (this is why no data is plotted for benchmarks with deadline slack less than 20%). In this set of experiments, the overhead values that have been considered for the dynamic case correspond actually to overheads by heuristics [20] and not the overheads incurred by exact methods, although in these experiments the reward values produced by the optimal solutions were considered. This means that, even in the optimistic case of an online algorithm that delivers exact solutions in a time frame similar to one of the existing heuristic methods (which naturally produce values of less quality than the exact ones), the quasi-static approach outperforms the dynamic one.

We have also measured the execution time of Algorithm 1, used for computing at design-time the set of V/O assignments. Fig. 13 shows the average execution time as a function of the number of tasks in the system, for different values of $N^{\max}$ (total number of assignments). It can be observed that the execution time is linear in the number of tasks and in the total number of assignments. The time needed for computing the set of assignments, though considerable, is affordable since Algorithm 1 is run offline.

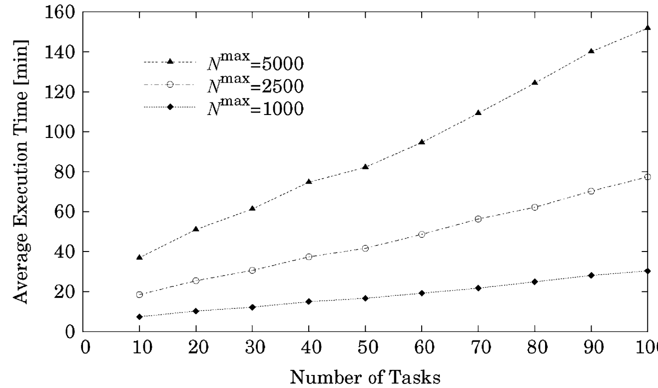



Fig. 13. Execution time of Algorithm 1.

In addition to the synthetic benchmarks previously discussed, we have also evaluated our approach by means of a real-life application, namely the navigation controller of an autonomous rover for exploring a remote place [11]. The rover is equipped, among others, with two cameras and a topographic map of the terrain. Based on the images captured by the cameras and the map, the rover must travel toward its destination avoiding nearby obstacles. This application includes a number of tasks described briefly as follows. A frame acquisition task captures images from the cameras. A position estimation task correlates the data from the captured images with the one from the topographic map in order to estimate the rover's current position. Using the estimated position and the topographic map, a global path planning task computes the path to the desired destination. Since there might be impassable obstacles along the global path, there is an object detection task for finding obstacles in the path of the rover and a local path planning task for adjusting accordingly the course in order to avoid those obstacles. A collision avoidance task checks the produced path to prevent the rover from damaging itself. Finally, a steering control task commands the motors, the direction, and speed of the rover.

For this application, the total reward is measured in terms of how fast the rover reaches its destination [11]. Rewards produced by the different tasks (all but the steering control task which has no optional part) contribute to the overall reward. For example, higher-resolution images by the frame acquisition task translates into a clearer characterization of the surroundings of the rover. This, in turn, implies a more accurate estimation of the location and, consequently, makes the rover get faster to its destination (that is, higher total reward). Similarly, running the global path planning task longer results in a better path which, again, implies reaching the desired destination faster. The other tasks make, in a similar manner, their individual contribution to the global reward, in such a way that the amount of computation allotted to each of them has a direct impact on how fast the destination is reached.

The navigation controller is activated periodically every 360 ms and tasks have a deadline equal to the period.[5] The energy budget per activation of the controller is 360 mJ (average power consumption 1 W) during the night and 540 mJ (average power 1.5 W) during the daytime [10].

[5]At its maximum speed of 10 km/h the rover travels in 360 ms a distance of 1 m, which is the maximum allowed without recomputing the path.

We use two memories, one for the assignments used during daytime and the other for the set used during the night (these two sets are different because the energy budget differs), and assume that $N^{\max} = 512$ assignments can be stored in each memory. We computed, for both cases, $E^{\max} = 360$ mJ and $E^{\max} = 540$ mJ, the sets of assignments using Algorithm 1. When compared to the respective static solutions, our quasi-static solution delivers rewards that are in average 3.8 times larger for the night case and 1.6 times larger for the day case. This means that a rover using the precomputed assignments can reach its destination faster than in the case of a static solution and, thus, explore a larger region under the same energy budget.

The significant difference between the night and day modes can be explained by the fact that, for more stringent energy constraints, fewer optional cycles can be accommodated by a static solution and, therefore, its reward is smaller. Thus, the relative difference between a quasi-static solution and the corresponding static one is significantly larger for systems with more stringent energy constraints.

## VII. Conclusion

We have addressed the problem of maximizing rewards for realtime systems with energy constraints, in the frame of the imprecise computation model. We have proposed a quasi-static approach, whose chief merit is the ability to exploit the dynamic slack at very low online overhead. This is possible because, in our quasi-static approach, a set of solutions are prepared and stored at design-time, leaving for runtime only the selection of one of them.

The number of assignments that can be stored is limited by the resources of the target system. Therefore, a careful selection of assignments is crucial because it has a large impact on the quality of the solution.

We considered that the voltage can continuously be varied. If only discrete voltages are supported, the approach can easily be adapted by using well-known techniques for obtaining the discrete voltage levels that replace the calculated ideal one [19].

We evaluated our approach through numerous synthetic benchmarks and a real-life application. We found that significant gains, up to four times, in terms of reward can be obtained by the quasi-static approach. We showed also that, due to its large online overheads, a dynamic approach performs poorly.

The dynamic slack can efficiently be exploited only if adversely high overheads are avoided, as done by our approach. The methods proposed in this paper succeed in exploiting the dynamic slack, yet having small online overhead, because the complex time- and energy-consuming parts of the computations are performed offline, at design-time, leaving for runtime only simple lookup and selection operations.

## Acknowledgment

The authors would like to thank A. Andrei for his help on the issues related to voltage-switching overheads and online computations.

## References

[1] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," in *Proc. DATE Conf.*, 2004, pp. 518–523.

[2] ——, "Quasi-static voltage scaling for energy minimization with time constraints," in *Proc. DATE Conf.*, 2005, pp. 514–519.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. Real-Time Syst. Symp.*, 2001, pp. 95–105.

[4] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "SystemC cosimulation and emulation of multiprocessor SoC designs," *IEEE Comput.*, vol. 36, no. 4, pp. 53–59, Apr. 2003.

[5] J. Cortadella, A. Kondratyev, L. Lavagno, and Y. Watanabe, "Quasi-static scheduling for concurrent architectures," in *Proc. Int. Conf. Appl. Concurrency Syst. Des.*, 2003, pp. 29–40.

[6] L. A. Cortés, "Verification and scheduling techniques for real-time embedded systems," Ph.D. dissertation, Dept. Comput. Inf. Sci., Linköping Univ., Linköping, Sweden, 2005.

[7] L. A. Cortés, P. Eles, and Z. Peng, "Quasi-static scheduling for real-time systems with hard and soft tasks," in *Proc. DATE Conf.*, 2004, pp. 1176–1181.

[8] L. A. Cortés, P. Eles, and Z. Peng, "Quasi-static assignment of voltages and optional cycles for maximizing rewards in real-time systems with energy constraints," in *Proc. DAC*, 2005, pp. 889–894.

[9] F. Gruian and K. Kuchcinski, "LEneS: Task scheduling for low-energy systems using variable supply voltage processors," in *Proc. ASP-DAC*, 2001, pp. 449–455.

[10] B. D. Hibbs, "Mars solar rover feasibility study," NASA, Washington, DC, Tech. Rep. NASA/CR 2001-210802, 2001.

[11] D. L. Hull, "An environment for imprecise computations," Ph.D. dissertation, Dept. Comput. Sci., Univ. Illinois, Urbana-Champaign, 2000.

[12] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," in *Proc. DAC*, 1997, pp. 697–702.

[13] C. M. Krishna and Y.-H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," *IEEE Trans. Comput.*, vol. 52, no. 12, pp. 1586–1593, Dec. 2003.

[14] J. W. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proc. IEEE*, vol. 82, no. 1, pp. 83–94, Jan. 1994.

[15] J. Luo and N. K. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," in *Proc. Int. Conf. VLSI Des.*, 2003, pp. 369–375.

[16] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for low power microprocessors under dynamic workloads," in *Proc. ICCAD*, 2002, pp. 721–725.

[17] *NEC Memories*, [Online]. Available: http://www.necel.com/memory/index_e.html

[18] Y. Nesterov and A. Nemirovski, *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia, PA: SIAM, 1994.

[19] T. Okuma, H. Yasuura, and T. Ishihara, "Software energy reduction techniques for variable-voltage processors," *IEEE Des. Test Comput.*, vol. 18, no. 2, pp. 31–41, Mar. 2001.

[20] C. Rusu, R. Melhem, and D. Mossé, "Maximizing rewards for real-time applications with energy constraints," *ACM Trans. Embedded Comput. Syst.*, vol. 2, no. 4, pp. 537–559, Nov. 2003.

[21] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, "Synthesis of embedded software using free-choice petri nets," in *Proc. DAC*, 1999, pp. 805–810.

[22] C.-S. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha, "Template-based real-time dwell scheduling with energy constraints," in *Proc. Real-Time Embedded Technol. Appl. Symp.*, 2003, pp. 19–27.

[23] W.-K. Shih, J. W. S. Liu, and J.-Y. Chung, "Fast algorithms for scheduling imprecise computations," in *Proc. Real-Time Syst. Symp.*, 1989, pp. 12–19.

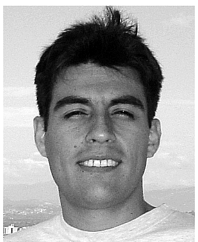
[24] D. Shin, S. Lee, and J. Kim, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Des. Test Comput.*, vol. 18, no. 2, pp. 20–30, Mar. 2001.

[25] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. DAC*, 1999, pp. 134–139.

[26] F.-S. Su and P.-A. Hsiung, "Extended quasi-static scheduling for formal synthesis and code generation of embedded software," in *Proc. CODES*, 2002, pp. 211–216.

[27] S. A. Vavasis, *Nonlinear Optimization: Complexity Issues*. New York: Oxford Univ. Press, 1991.

[28] P. Yang and F. Catthoor, "Pareto-optimization-based run-time task scheduling for embedded systems," in *Proc. CODES ISSS*, 2003, pp. 120–125.
[29] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. Symp. Foundations Comput. Sci.*, 1995, pp. 374–382.
[30] H.-S. Yun and J. Kim, "Reward-based voltage scheduling for fixed-priority hard real-time systems," in *Proc. Int. Workshop on Power-Aware Real-Time Comput.*, 2004, pp. 1–4.
[31] Y. Zhang, X. S. Hu, and D. Z. Chen, "Energy minimization of real-time tasks on variable voltage processors with transition overheads," in *Proc. ASP-DAC*, 2003, pp. 65–70.



**Luis Alejandro Cortés** (M'00) received the B.Sc. degree from the National University of Colombia, Bogotá, Columbia, in 1995, the M.Sc. degree from the University of Los Andes, Bogotá, Columbia, in 1998, both in electrical engineering, and the Ph.D. degree in computer science from Linköping University, Linköping, Sweden, in 2005. His Ph.D. dissertation "Verification and Scheduling Techniques for Real-Time Embedded Systems," dealt with the design and verification of embedded systems, with special emphasis on formal methods and real-time aspects.

He is currently with Volvo Truck Corporation, Gothenburg, Sweden, where he is developing future electronic architecture to be shared by different truck brands within the Volvo Group. His research interests include embedded systems design, real-time systems, and energy-aware design, with focus on automotive applications.

**Petru Eles** (M'99) received the Ph.D. degree in computer science from the Politehnica University of Bucharest, Bucharest, Romania, in 1993.

He is currently a Professor with the Department of Computer and Information Science at Linköping University, Linköping, Sweden. His research interests include embedded systems design, hardware-software codesign, real-time systems, system specification and testing, and computer-aided design (CAD) for digital systems. He has published extensively in these areas and coauthored several books, such as *System Synthesis with VHDL* (Kluwer, 1997), *System-Level Design Techniques for Energy-Efficient Embedded Systems* (Kluwer, 2003), and *Analysis and Synthesis of Distributed Real-Time Embedded Systems* (Kluwer, 2004).

Dr. Eles is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and of the *IEE Proceedings—Computers and Digital Techniques*. He has served as a Program Committee member for numerous international conferences in the areas of Design Automation, Embedded Systems, and Real-Time Systems, and as a TPC chair and General chair of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. He was a corecipient of the Best Paper Awards at the European Design Automation Conference in 1992 and 1994, and at the Design Automation and Test in Europe Conference in 2005, and of the Best Presentation Award at the 2003 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. He has served as an IEEE Circuits and Systems Society Distinguished Lecturer for 2004 and 2005. He is a member of the ACM.

**Zebo Peng** (M'91–SM'02) received the B.Sc. degree in computer engineering from the South China Institute of Technology, Guangzhou, China, in 1982, and the Licentiate of Engineering and Ph.D. degrees in computer science from Linköping University, Linköping, Sweden, in 1985 and 1987, respectively.

Currently, he is a Full Professor of Computer Systems, Director of the Embedded Systems Laboratory, and Chairman of the Division for Software and Systems in the Department of Computer Science, Linköping University. His research interests include design and test of embedded systems, electronic design automation, SoC testing, design for testability, hardware/software codesign, and realtime systems. He has published 200 technical papers in these areas and coauthored the books *System Synthesis with VHDL* (Kluwer, 1997), *Analysis and Synthesis of Distributed Real-Time Embedded Systems* (Kluwer, 2004), and *System-level Test and Validation of Hardware/Software Systems* (Springer, 2005).

Prof. Peng has served on the program committee of a dozen international conferences and workshops, including ATS, ASP-DAC, DATE, DDECS, DFT, ETS, ITSW, MEMOCODE, and VLSI-SOC. He was the General Chair of the 6th IEEE European Test Workshop (ETW'01), the Program Chair of the 7th IEEE Design and Diagnostics of Electronic Circuits & Systems Workshop (DDECS'04), and the Test Track Chair of the 2006 Design Automation and Test in Europe Conference (DATE'06). He is the Program Chair of the 12th IEEE European Test Symposium (ETS'07) and the Vice Program Chair of DATE'07. He currently serves as the Chair of the IEEE European Test Technology Technical Council (ETTTC). He was corecipient of two Best Paper Awards at the European Design Automation Conferences (1992 and 1994), a Best Paper Award at the IEEE Asian Test Symposium (2002), a Best Paper Award at the Design Automation and Test in Europe Conference (2005), and a Best Presentation Award at the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (2003).