in *Proc. Intl. Conference on Real-Time and Embedded Computing Systems and Applications*, 2006, pp. 279-286.

# A Quasi-Static Approach to Minimizing Energy Consumption in Real-Time Systems under Reward Constraints

Luis Alejandro Cortés [1] alejandro.cortes@volvo.com
[1] Volvo Truck Corporation Gothenburg, Sweden

Petru Eles [2] petel@ida.liu.se
Zebo Peng [2] zebpe@ida.liu.se
[2] Linköping University Linköping, Sweden

## Abstract

*In some real-time applications, it is desirable to trade off precision for timeliness. For such systems, considered typically under the Imprecise Computation model, a function assigns reward to the application depending on the amount of computation allotted to it. Also, many such applications run on battery-powered devices where the energy consumption is of utmost importance. We address in this paper the problem of energy minimization for Imprecise-Computation systems that have reward and time constraints. We propose a Quasi-Static (QS) approach that exploits, with low on-line overhead, the dynamic slack that arises from variations in the actual number of execution cycles: first, at design-time, a set of solutions are computed and stored (off-line phase); second, the selection among the precomputed assignments is left for run-time, based on actual values of time and reward (on-line phase).*

## 1 Introduction

Power and energy consumption have become quite important design considerations. Dynamic Voltage Scaling (DVS) techniques [9] are a well-known approach for reducing the energy consumption in real-time systems. By lowering the supply voltage quadratic savings in energy consumption can be achieved while performance is degraded in approximately linear fashion. At the same time, for certain real-time applications approximate but timely results are acceptable, for example, fuzzy images in time are often preferable to perfect images too late. Imprecise Computation (IC) techniques [5] have been used for studying such systems. Tasks are composed of mandatory and optional parts, both of which must be finished by the deadline, although the optional part can be left incomplete at the expense of the quality of results (a function assigns reward depending on the amount of computation allotted to the optional part).

On the one hand, DVS techniques, which allow the trade-off between energy consumption and performance, have mainly been applied to hard real-time systems (no reward aspect considered). On the other hand, IC approaches, which make it possible to trade off precision for timeliness, have until now disregarded the energy aspects. Rusu *et al.* [10] proposed an approach in which reward, energy, and deadlines are considered in the same framework. The problem is to maximize the total reward without exceeding the energy budget or the deadlines. This approach solves statically the optimization problem and consequently considers only worst cases. A similar problem was discussed by Cortés *et al.* [4] but, as opposed to [10], the dynamic slack, caused by tasks completing earlier than in the worst case, is exploited by using a QS approach.

In this paper we also deal with real-time systems for which it is possible to trade off precision for timeliness as well as energy consumption for performance. The problem addressed in this paper (somehow a mirror problem to the one in [4]) is to minimize the energy consumption subject to a minimum total-reward constraint and deadlines. We aim at finding the voltage levels at which each task runs and its number of optional cycles such that the objective function is optimized and the constraints satisfied.

A static solution implies finding one Voltage/Optional-cycles (V/O) assignment; it is pessimistic because actual execution times are typically far off from worst-case values. A dynamic solution implies recomputing, every time a task completes, a V/O assignment; although it can exploit the dynamic slack, the on-line overhead is too high to make the dynamic solution applicable in practice. We propose a *quasi-static* approach composed of two steps: first, at design time, we compute a set of V/O assignments (off-line phase); second, at run time, one of the precomputed V/O assignments is selected based on actual values of time and accumulated reward (on-line phase).

To our knowledge this is the first paper that considers the problem of energy minimization in the frame of IC systems. A chief merit of our approach is its ability to effectively exploit the dynamic slack at very low on-line overhead.

## 2 Preliminaries

### 2.1 Task and Architectural Models

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. 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 $M_i^{\text{bc}}$ and the worst-case number of cycles $M_i^{\text{wc}}$, that is, $M_i^{\text{bc}} \leq M_i \leq M_i^{\text{wc}}$. The expected number of mandatory cycles of a task $T_i$ is denoted $M_i^{\text{e}}$. 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 argument the number of optional cycles $O_i$ assigned to $T_i$; we assume that $R_i(0) = 0$. We consider non-decreasing concave[1] reward functions as they capture the particularities of most real-life applications [10]. Also, as detailed in Section 4, 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 $R = \sum_{T_i \in \mathbf{T}} R_i(O_i)$. The reward produced up to the completion of task $T_i$ is denoted $RP_i$ ($RP_i = \sum_{j=1}^{i} R_j(O_j)$). We

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

consider a reward constraint, denoted $R^{\min}$, that gives the lower bound of the total reward that must be produced.

We consider that tasks are non-preemptable 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 off-line, that respects the data dependencies and according to an EDF (Earliest Deadline First) policy. For non-preemptable tasks with equal release time and running on a single processor, EDF gives the optimal execution order [3][2]. $T_i$ denotes the $i$-th task in this sequence.

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 be adapted by using well-known techniques for determining the discrete voltage levels that replace the calculated continuous one [9].

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

### 2.2 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 [6]:

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

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 [6]:

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

where $C_r$ is the capacitance of the power rail. We also consider, for the QS solution, the energy overhead $\mathcal{E}_i^{sel}$ 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 $\mathcal{E}_i^{sel}$ is a constant value. Also, this value is the same for all tasks ($\mathcal{E}_i^{sel} = \mathcal{E}^{sel}$, for $1 \leq i \leq n$). For consistency reasons we keep the index $i$ in the notation of the selection overhead $\mathcal{E}_i^{sel}$. The energy overhead caused by on-line operations is denoted $\mathcal{E}_i^{dyn}$. In a QS solution the on-line overhead is just the selection overhead ($\mathcal{E}_i^{dyn} = \mathcal{E}_i^{sel}$) [4].

The execution time of a task $T_i$ executing $M_i + O_i$ cycles at supply voltage $V_i$ is [6]:

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

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 $V_{th}$ is the threshold voltage. The time overhead, when switching from $V_i$ to $V_j$, is given by the following expression [1]:

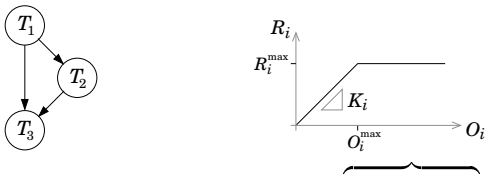
$$\delta_{i,j}^{\Delta V} = p|V_i - V_j| \tag{4}$$

where $p$ is a constant. The time overhead for looking up and selecting one V/O assignment in the QS approach is denoted $\delta_i^{sel}$ 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. $\delta_i = \delta_{i-1,i}^{\Delta V} + \delta_i^{dyn}$ where $\delta_i^{dyn}$ is the on-line overhead. Note that in a QS solution this on-line overhead is just the lookup and selection time, that is, $\delta_i^{dyn} = \delta_i^{sel}$.

## 3 Motivational Example

Before going into the precise formulation and the details of the solution, we consider in this section the example shown in Fig. 1. We assume non-decreasing reward functions of the form $R_i(O_i) = K_i O_i$, $O_i \leq O_i^{\max}$ as well as a reward constraint $R^{\min} = 8$. As explained in Subsection 2.1, tasks run according to the schedule $T_1 T_2 T_3$, fixed off-line in conformity to an EDF policy. We consider a processor that permits continuous voltage scaling in the range 0.6-1.8 V. For the sake of clarity, in this example we assume that transition overheads are zero.




| Task | $M_i^{\text{bc}}$ | $M_i^{\text{e}}$ | $M_i^{\text{wc}}$ | $C_i$ [nF] | $d_i$ [$\mu$s] | $K_i$ | $R_i^{\max}$ |
|---|---|---|---|---|---|---|---|
| $T_1$ | 30000 | 60000 | 80000 | 0.6 | 200 | 0.00012 | 3.60 |
| $T_2$ | 20000 | 40000 | 50000 | 1.0 | 300 | 0.00021 | 5.25 |
| $T_3$ | 10000 | 50000 | 90000 | 0.8 | 550 | 0.00014 | 5.60 |

Fig. 1. Motivational example

The optimal static V/O assignment for this example is given by Table 1. 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 minimize the energy consumption, while guaranteeing that deadlines are met and the reward constraint is satisfied.

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.744 | 14 |
| $T_2$ | 1.488 | 23864 |
| $T_3$ | 1.471 | 21342 |

Table 1. Optimal static V/O assignment

The V/O assignment given by Table 1 is optimal in the static sense. It is the best possible that can be obtained off-line 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. This point is illustrated by the following situation. The first task starts executing at $V_1 = 1.744$ V, as required by the static assignment. Assume that $T_1$ executes $M_1 = 40000$ (instead of $M_1^{\text{wc}} = 80000$) mandatory cycles and then its assigned $O_1 = 14$ optional cycles. At this point, knowing that $T_1$ has

[2] By optimal in this context we mean the task execution order that, among all feasible orders, admits the V/O assignment for which the lowest total energy can be achieved. We have demonstrated in [3] that an EDF execution order is the one that least constraints the space of V/O solutions and henceforth optimal in the above sense.

completed at $t_1 = \tau_1 = 68.54$ $\mu$s and that the reward produced is $RP_1 = R_1 = 0.00168$, a new V/O assignment can accordingly be computed for the remaining tasks aiming at obtaining the minimum total energy for the new conditions. We consider, for the moment, the ideal case in which such an on-line 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 $V_2 = 1.257$ V and $O_2 = 24993$. Then $T_2$ runs at $V_2 = 1.257$ V and let us assume that it executes $M_2 = 30000$ (instead of $M_2^{\text{wc}} = 50000$) mandatory cycles and then its newly assigned $O_2 = 24993$ optional cycles. At this point, the completion time is $t_2 = \tau_1 + \tau_2 = 230.3$ $\mu$s and the reward so far produced is $RP_2 = R_1 + R_2 = 5.25021$. Again, a new assignment can be computed taking into account the information about completion time and produced reward. This new assignment gives $V_3 = 1.264$ V and $O_3 = 19644$.

For such a situation, in which $M_1 = 40000$, $M_2 = 30000$, and $M_3 = 80000$, the V/O assignment computed dynamically (considering $\delta^{dyn} = 0$) is summarized in Table 1(a). According to this assignment and considering $M_1 = 40000$, $M_2 = 30000$, $M_3 = 80000$, the total energy (assuming also $\mathcal{E}^{dyn} = 0$) is $E^{dyn^{ideal}} = 287.27$ $\mu$J. In reality, however, the on-line time and memory overheads caused by computing new assignments are not negligible. When considering, for example, the overhead $\delta^{dyn} = 40$ $\mu$s the V/O assignment computed dynamically is evidently different, as given by Table 1(b). This assignment makes the total energy consumed (assuming that the on-line computation of V/O assignments takes $\mathcal{E}^{dyn} = 35$ $\mu$J) be $E^{dyn^{real}} = 405.92$ $\mu$J. The values of $\delta^{dyn}$ and $\mathcal{E}^{dyn}$ 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 on-line heuristics, which produce approximate results, have long execution times. This means that a dynamic V/O scheduler might produce solutions that are actually inferior to the static one (in terms of total energy consumed) or, even worse, a dynamic V/O scheduler might not be able to fulfill the given time and reward constraints.

(a) $\delta^{dyn} = 0$

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.744 | 14 |
| $T_2$ | 1.257 | 24993 |
| $T_3$ | 1.264 | 19644 |

(b) $\delta^{dyn} = 40$ $\mu$s

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.744 | 14 |
| $T_2$ | 1.363 | 24998 |
| $T_3$ | 1.420 | 19634 |

Table 2. Dynamic V/O assignments (for $M_1 = 40000$, $M_2 = 30000$, $M_3 = 80000$)

Observe that for the situation of number of mandatory cycles considered above ($M_1 = 40000$, $M_2 = 30000$, $M_3 = 80000$), the total energy consumed when using the static assignment of Table 1 is $E^{st} = 367.72$ $\mu$J while the ideal dynamic assignment of Table 1(a) gives $E^{dyn^{ideal}} = 287.27$ $\mu$J, that is, energy savings of 22%. This shows that important energy savings might be achieved by exploiting the dynamic slack. At the same time, the dynamic assignment of Table 1(b), which takes into account the on-line overheads, gives a total energy $E^{dyn^{real}} = 405.92$ $\mu$J, or 10% *more* than the static assignment; this inferior result is caused by the time and energy on-line overheads. The above figures illustrate that the dynamic slack can efficiently be exploited only if methods with low on-line overheads are used. In our QS approach we compute at design-time a number of V/O assignments, which are selected at run-time by the so-called QS V/O scheduler (at very low overhead).

We can define, for instance, a QS set of assignments for the example discussed in this subsection, as given by Table 3. Although this set was obtained by using the particular solution we propose in Section 5 (in which the number of optional cycles is "frozen" as explained later), it illustrates well the essence of the QS approach. These assignments were computed considering the selection overheads $\delta^{sel} = 0.3$ $\mu$s and $\mathcal{E}^{sel} = 0.3$ $\mu$J. At run-time, upon completion of each task, $V_i$ and $O_i$ are selected from the precomputed set according to the given condition.

| Task | Condition | $V_i$ [V] | $O_i$ |
|---|---|---|---|
| $T_1$ | — | 1.744 | 14 |
| $T_2$ | if $t_1 \leq 84$ $\mu$s | 1.285 | 23864 |
| | else if $t_1 \leq 110$ $\mu$s | 1.361 | 23864 |
| | else | 1.488 | 23864 |
| $T_3$ | if $t_2 \leq 180$ $\mu$s | 1.176 | 21342 |
| | else if $t_2 \leq 250$ $\mu$s | 1.321 | 21342 |
| | else | 1.471 | 21342 |

Table 3. Precomputed set of V/O assignments

For the situation $M_1 = 40000$, $M_2 = 30000$, $M_3 = 80000$ and the set given by Table 3, the QS V/O scheduler would do as follows. Task $T_1$ is run at $V_1 = 1.744$ V and is allotted $O_1 = 14$ optional cycles. Since, when completing $T_1$, $t_1 = \tau_1 = 68.54 \leq 84$ $\mu$s, $V_2 = 1.285/O_2 = 23864$ is selected by the QS V/O scheduler. Task $T_2$ runs under this assignment so that, when it finishes, $t_2 = \tau_1 + \delta_2^{sel} + \tau_2 = 220.83$ $\mu$s. Then $V_3 = 1.321/O_3 = 21342$ is selected and task $T_3$ is executed accordingly. Table 4 summarizes the selected assignment. The energy consumed, when using this V/O assignment is $E^{qs} = 304.04$ $\mu$J (compare to $E^{dyn^{ideal}} = 287.27$ $\mu$J, $E^{dyn^{real}} = 405.92$ $\mu$J, and $E^{st} = 367.72$ $\mu$J). Two important facts can be noted from the example: first, the QS solution *qs* outperforms clearly the dynamic one $dyn^{real}$ because of the large overheads of the latter; second, the QS solution *qs* is not far from the ideal case of a dynamic V/O scheduler $dyn^{ideal}$ with zero overheads.

| Task | $V_i$ [V] | $O_i$ |
|---|---|---|
| $T_1$ | 1.744 | 14 |
| $T_2$ | 1.285 | 23864 |
| $T_3$ | 1.321 | 21342 |

Table 4. QS V/O assignment (for $M_1 = 40000$, $M_2 = 30000$, $M_3 = 80000$) selected from the set of Table 3

## 4 Problem Formulation

In this paper—under the framework of the Imprecise Computation model—we discuss the problem of minimizing the energy consumption considering that there is a minimum total reward that must be delivered by the system as well as time constraints in the form of deadlines that must be met.

In what follows we present the precise formulation of related problems as well as 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.

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

$$\text{minimize} \sum_{i=1}^{n} \Big( \underbrace{C_r(V_{i-1} - V_i)^2}_{\mathcal{E}^{\Delta V}_{i-1,i}} + \underbrace{C_i V_i^2 (M_i^{\mathrm{e}} + O_i)}_{E_i^{\mathrm{e}}} \Big) \tag{5}$$

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

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

$$\sum_{i=1}^{n} R_i(O_i) \geq R^{\min} \tag{8}$$

The above formulation can be explained as follows. The objective function to be minimized is the total energy, which is the sum of the voltage-switching energies $\mathcal{E}^{\Delta V}_{i-1,i}$ and the energy $E_i^{\mathrm{e}}$ consumed by each task (Eq. (5)). The voltage $V_i$ for each task $T_i$ must be in the range $[V^{\min}, V^{\max}]$ (Eq. (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 $\tau_i$, and tasks must complete before their deadlines $d_i$ (Eq. (7)); note that the worst-case number of mandatory cycles has to be assumed in order to guarantee the deadlines. The total reward has to be at least $R^{\min}$ (Eq. (8)).

When solving the above problem, for tractability reasons, we consider $O_i$ as a continuous variable and then 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) [3]. It can also be noted that in the above problem the objective as well as the constraint functions are convex[3]. Therefore we have a *convex non-linear programming* (NLP) formulation [11] and hence the problem can be solved using polynomial-time methods [8].

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 reward produced up to the completion of $T_c$ is $RP_c$ and the completion time of $T_c$ is $t_c$).

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

$$\text{minimize} \sum_{i=c+1}^{n} \Big( \mathcal{E}_i^{dyn} + \mathcal{E}^{\Delta V}_{i-1,i} + \underbrace{C_i V_i^2 (M_i^{\mathrm{e}} + O_i)}_{E_i^{\mathrm{e}}} \Big) \tag{9}$$

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

$$s_{i+1} = t_i = s_i + \delta_i^{dyn} + \delta^{\Delta V}_{i-1,i} + \underbrace{k \frac{V_i}{(V_i - V_{th})^{\alpha}} (M_i^{\mathrm{wc}} + O_i)}_{\tau_i^{\mathrm{wc}}} \leq d_i \tag{11}$$

$$\sum_{i=c+1}^{n} R_i(O_i) \geq (R^{\min} - RP_c) \tag{12}$$

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

The problem solved by the above dynamic V/O scheduler corresponds to an instance of the static V/O assignment problem (for $c+1 \leq i \leq n$ and taking into account $t_c$ and $RP_c$), but considering $\delta_i^{dyn}$ and $\mathcal{E}_i^{dyn}$. However, a *speculative* version of the dynamic V/O scheduler can be formulated as follows. Such a dynamic speculative V/O scheduler produces better results than its non-speculative counterpart, as shown by the experimental results of Section 6.

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

DYNAMIC SPECULATIVE V/O SCHEDULER: The following is the problem that a dynamic speculative 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 reward produced up to the completion of $T_c$ is $RP_c$ and the completion time of $T_c$ is $t_c$).

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

$$\text{minimize} \sum_{i=c+1}^{n} \Big( \mathcal{E}_i^{dyn} + \mathcal{E}^{\Delta V}_{i-1,i} + \underbrace{C_i V_i^2 (M_i^{\mathrm{e}} + O_i)}_{E_i^{\mathrm{e}}} \Big) \tag{13}$$

$$\text{subject to } V^{\min} \leq V_i \leq V^{\max} \tag{14}$$

$$s_{i+1} = t_i = s_i + \delta_i^{dyn} + \delta^{\Delta V}_{i-1,i} + \underbrace{k \frac{V_i}{(V_i - V_{th})^{\alpha}} (M_i^{\mathrm{e}} + O_i)}_{\tau_i^{\mathrm{e}}} \leq d_i \tag{15}$$

$$\sum_{i=c+1}^{n} R_i(O_i) \geq (R^{\min} - RP_c) \tag{16}$$

$$s'_{i+1} = t'_i = s'_i + \delta_i^{dyn} + \delta^{\Delta V}_{i-1,i} + \tau'_i \leq d_i \tag{17}$$

$$\tau'_i = \begin{cases} k \frac{V_i}{(V_i - V_{th})^{\alpha}} (M_i^{\mathrm{wc}} + O_i) & \text{if } i = c+1 \\ k \frac{V^{\max}}{(V^{\max} - V_{th})^{\alpha}} (M_i^{\mathrm{wc}} + O_i) & \text{if } i > c+1 \end{cases} \tag{18}$$

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

Eqs. (13)-(16) are basically the same as Eqs. (9)-(12) except that the expected number of mandatory cycles $M_i^{\mathrm{e}}$ is used instead of the worst-case number of mandatory cycles $M_i^{\mathrm{wc}}$ in the constraint corresponding to the deadlines (see Eqs. (11) and (15)). The constraint given by Eq. (15) does not guarantee by itself the satisfaction of deadlines because if the actual number of mandatory cycles is larger than $M_i^{\mathrm{e}}$, deadline violations might arise. Therefore an additional constraint, as given by Eqs. (17) and (18), is introduced. It expresses that: the next task $T_{c+1}$, running at $V_{c+1}$, must meet its deadline ($T_{c+1}$ will run at the computed $V_{c+1}$); the other tasks $T_i$, $c+1 < i \leq n$, running at $V^{\max}$, must also meet the deadlines (the other tasks $T_i$ might run at a voltage different from the value $V_i$ computed in the current iteration, because solutions obtained upon completion of future tasks might produce different values). Guaranteeing the deadlines in this way is possible because new assignments are similarly recomputed every time a task finishes.

The dynamic speculative V/O scheduler presented above solves the V/O assignment problem speculating that tasks will execute their expected number of mandatory cycles but leaving enough room for increasing the voltage so that future tasks, if needed, run faster and thus meet the deadlines. We consider that the energy $E^{ideal}$ consumed by a system, when the V/O assignments are computed by such a dynamic speculative V/O scheduler in the ideal case $\delta_i^{dyn} = 0$ and $\mathcal{E}_i^{dyn} = 0$, is the lower bound on the total energy that can practically be achieved without knowing in advance how many mandatory cycles tasks will execute and without accepting risks regarding deadline of reward 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 off-line a number of V/O assignments, one of which is to be selected by the *QS V/O scheduler*.

Upon finishing a task $T_c$, the QS V/O scheduler checks the completion time $t_c$ and the reward $RP_c$ produced up to completion of $T_c$, and looks up an assignment in $\mathrm{LUT}_c$. From the lookup table $\mathrm{LUT}_c$ the QS V/O scheduler gets the point $(t'_c, RP'_c)$, which is the closest to $(t_c, RP_c)$ such that

$t_c \leq t'_c$ and $RP_c \geq RP'_c$, and selects $V'/O'$ corresponding to $(t'_c, RP'_c)$. The goal is to make the system consume as little energy as possible, when using the assignments selected by the QS V/O scheduler. The problem we discuss in the rest of the paper is:

SET OF V/O ASSIGNMENTS: Find a set of $N$ assignments such that: $N \leq N^{\max}$; the V/O assignment selected by the QS V/O scheduler guarantees the deadlines ($s_i + \delta_i^{sel} + \delta_{i-1,i}^{\Delta V} + \tau_i = t_i \leq d_i$) and the reward constraint ($\sum_{i=1}^{n} R_i(O_i) \geq R^{\min}$), and so that the total energy $E^{qs}$ is minimal.

As discussed in Section 5, for a task $T_i$, potentially there exist infinitely many possible values for $t_i$ and $RP_i$. Therefore, in order to approach the theoretical limit $E^{ideal}$, it would be needed to compute an infinite number of V/O assignments, one for each $(t_i, RP_i)$. The problem is thus how to select at most $N^{\max}$ points in this infinite space such that the energy consumed, when using the respective V/O assignments, is as close as possible to $E^{ideal}$.

## 5 Computing the Set of V/O Assignments

For each task $T_i$, there is a *time-reward* space of possible values of completion time $t_i$ and reward $RP_i$ produced up to completion of $T_i$, as depicted in Fig. 2. Each point in this space defines a V/O assignment for the next task $T_{i+1}$: if $T_i$ finished at $t^a$ and the produced reward was $RP^a$, the assignment for the next task would be $V_{i+1} = V^a / O_{i+1} = O^a$ (that is, $T_{i+1}$ would run at $V^a$ and execute $O^a$ optional cycles). The values $V^a$ and $O^a$ are those that an ideal dynamic speculative V/O scheduler would give in the case $t_i = t^a$, $RP_i = RP^a$ (recall that we aim at matching $E^{ideal}$ which is the energy consumed when the V/O assignments produced by such an ideal dynamic speculative V/O scheduler are used). Different points $(t_i, RP_i)$ define different V/O assignments as shown in Fig. 2. Note also that for a given value $t_i$ there might be different valid values of $RP_i$, and this is due to the fact that different previous V/O assignments can lead to the same $t_i$ but still different $RP_i$.

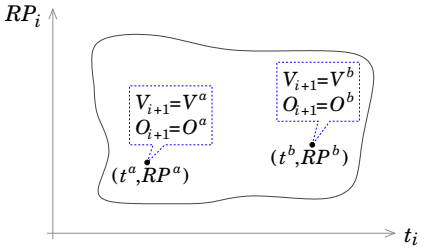



Fig. 2. Time-reward space

In order to select $N_i$ points in the $t_i$-$RP_i$ space and accordingly compute the set of $N_i$ assignments, it is first needed to determine the boundaries of this space (for each task $T_i$). $N_i$ 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. The boundaries of the $t_i$-$RP_i$ space can be obtained by computing the extreme values of $t_i$ (earliest and latest completion times) and of $RP_i$ (minimum and maximum reward produced up to task $T_i$), considering $V^{\min}$, $V^{\max}$, $M_j^{\text{bc}}$, $M_j^{\text{wc}}$, and $O_j^{\max}$, $1 \leq j \leq i$. The maximum produced reward is $RP_i^{\max} = \sum_{j=1}^{i} R_j(O_j^{\max})$ and the minimum reward is simply $RP_i^{\min} = \sum_{j=1}^{i} R_j(0) = 0$. The maximum completion time $t_i^{\max}$ occurs when each task $T_j$ executes $M_j^{\text{wc}} + O_j^{\max}$ cycles at $V^{\min}$, while $t_i^{\min}$ happens when each task $T_j$ executes $M_j^{\text{bc}}$ cycles at $V^{\max}$. The intervals $[t_i^{\min}, t_i^{\max}]$ and $[0, RP_i^{\max}]$ bound the time-reward space as shown in Fig. 3.

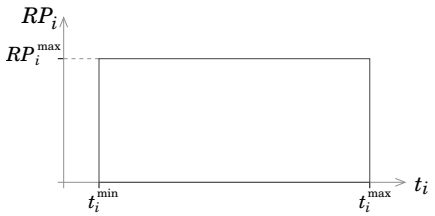



Fig. 3. Boundaries of the time-reward space

A generic characterization of the time-reward space is not possible because reward functions vary from task to task as well as from system to system: we cannot derive a general expression that relates the reward $R_i$ with the execution time $\tau_i$ and hence a characterization of the $t_i$-$RP_i$ space is not possible.

One alternative for selecting points in the time-reward space would be to consider a mesh-like configuration, in which the space is divided in rectangular areas and each area is covered by one point (the lower-right corner covers the rectangle) as depicted in Fig. 4. The drawback of this approach is twofold: first, the boundaries in Fig. 3 define a time-reward space that include points that cannot happen, for example, the point $(t_i^{\min}, RP_i^{\max})$ is not feasible because $t_i^{\min}$ occurs when no optional cycles are executed whereas $RP_i^{\max}$ requires all tasks $T_j$ executing $O_j^{\max}$ optional cycles; second, the number of required points for covering the space is a *quadratic* function of the granularity of the mesh, which means that too many points might be necessary for achieving an acceptable granularity.

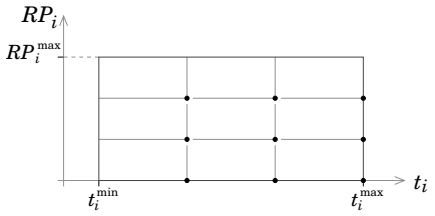



Fig. 4. Points in a mesh configuration

We have opted for a solution where we "freeze" the assigned optional cycles, that is, for each task $T_i$ we fix $O_i$ to a value $\overline{O}_i$ computed off-line. Thus, in the solution proposed in this paper, for any activation of the system, $T_i$ will invariably execute $\overline{O}_i$ optional cycles. In this way, we transform the original problem into a classical voltage-scaling problem with deadlines since the only variables now are $V_i$. This means that we reduce the bidimensional time-reward space into a one-dimension space (time is now the only dimension). This approach gives very good results as shown by the experimental evaluation presented in Section 6.

By freezing the optional cycles $O_i$, although the space of solutions is constrained, good results can be achieved because the only variable that affects the reward dimension is $O_i$ and, for this problem, the reward is a constraint. Thus, if the optional cycles are frozen in such a way that the reward constraint is satisfied, there is still enough room for exploiting the dynamic slack caused by tasks executing less mandatory

cycles than in the worst case: there is no gain by running more optional cycles and accordingly producing more reward than required by the reward constraint.

The way we obtain the fixed values $\overline{O}_i$ is the following. We consider the instance of the problem—as formulated by Eqs. (13)-(18)—that the dynamic speculative V/O scheduler solves at the very beginning, before any task is executed ($c = 0$). The solution gives particular values of $V_i$ and $O_i$, $1 \leq i \leq n$. For each task, the number of optional cycles given by this solution is taken as the fixed value $\overline{O}_i$ in our approach.

Once the number of optional cycles has been fixed to $\overline{O}_i$, the only variables are $V_i$ and the problem becomes that of *voltage scaling for energy minimization with time constraints*. For the sake of completeness, we include below its formulation. The reward constraint disappears from the formulation because, by fixing the optional cycles as explained above, it is guaranteed that the total reward will be at least $R^{\min}$.

DYNAMIC VOLTAGE SCHEDULER: The following is the problem that a dynamic voltage scheduler must solve every time a task $T_c$ completes. It is considered that tasks $T_1, \ldots, T_c$ have already completed (the completion time of $T_c$ is $t_c$).

Find $V_i$, for $c+1 \leq i \leq n$, that

$$\text{minimize} \sum_{i=c+1}^{n} \Big(\mathcal{E}_i^{dyn} + \mathcal{E}_{i-1,i}^{\Delta V} + \underbrace{C_i V_i^2 (M_i^{\mathrm{e}} + \overline{O}_i)}_{E_i^{\mathrm{e}}}\Big) \tag{19}$$

$$\text{subject to } V^{\min} \leq V_i \leq V^{\max} \tag{20}$$

$$s_{i+1} = t_i = s_i + \delta_i^{dyn} + \delta_{i-1,i}^{\Delta V} + \underbrace{k \frac{V_i}{(V_i - V_{th})^{\alpha}} (M_i^{\mathrm{e}} + \overline{O}_i)}_{\tau_i^{\mathrm{e}}} \leq d_i \tag{21}$$

$$s'_{i+1} = t'_i = s'_i + \delta_i^{dyn} + \delta_{i-1,i}^{\Delta V} + \tau'_i \leq d_i \tag{22}$$

$$\tau'_i = \begin{cases} k \dfrac{V_i}{(V_i - V_{th})^{\alpha}} (M_i^{\mathrm{wc}} + \overline{O}_i) & \text{if } i = c+1 \\ k \dfrac{V^{\max}}{(V^{\max} - V_{th})^{\alpha}} (M_i^{\mathrm{wc}} + \overline{O}_i) & \text{if } i > c+1 \end{cases} \tag{23}$$

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

In our QS approach, once the number of assigned optional cycles has been "frozen" to $\overline{O}_i$, we take $N_i$ points $t_i^j$ along the interval $[\bar{t}_i^{\min}, \bar{t}_i^{\max}]$; the earliest completion time $\bar{t}_i^{\min}$ occurs when each of the previous tasks $T_j$ (inclusive $T_i$) execute their minimum number of cycles $M_j^{\mathrm{bc}}$ and $\overline{O}_j$ optional cycles at maximum voltage $V^{\max}$, while $\bar{t}_i^{\max}$ occurs when each task $T_j$ executes $M_j^{\mathrm{wc}} + \overline{O}_j$ cycles at $V^{\min}$. Then we compute and store the respective voltage settings $V_{i+1}^j$ that minimize the total energy when $T_i$ completes at $t_i^j$, according to the formulation given by Eqs. (19)-(23). It should be noted that for the computation of the voltage $V_{i+1}^j$, the time and energy overheads $\delta_i^{dyn} = \delta_i^{sel}$ and $\mathcal{E}_i^{dyn} = \mathcal{E}_i^{sel}$ (needed for selecting voltages at run-time) are taken into account.

Each one of the points, together with its corresponding assignment, covers a region as indicated in Fig. 5. The QS scheduler selects one of the stored assignments based on the actual completion time. If, for example, task $T_i$ completes at $t'$, $t^a < t' \leq t^b$, the QS V/O scheduler will select the precomputed assignment $V^b/\overline{O}$. Note that we have included in Fig. 5 the optional cycles $\overline{O}$ for the sake of making clearer the nature of our approach. However, in practice, there is no need to store the number of optional cycles in the lookup tables $\mathrm{LUT}_i$ since, once these are "frozen", task $T_i$ will invariably execute $\overline{O}_i$ optional cycles.

The pseudocode corresponding to the computations performed off-line for obtaining the set of 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, it is more efficient to distribute the assignments according to the size of the interval $[\bar{t}_i^{\min}, \bar{t}_i^{\max}]$, in such a way that the lookup tables of tasks with larger intervals get more points.

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

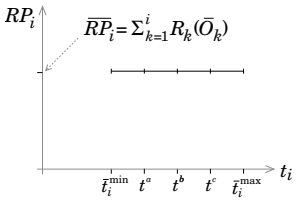



Fig. 5. Illustration of a lookup table

Then we compute the solution of the problem formulated by Eqs. (13)-(18) for $c = 0$ and we "freeze" the number of optional cycles according to this solution (line 2). Since the assignment $V_1$ is invariably the same (task $T_1$ runs always at the same voltage level), this is the only one stored for the first task (line 3). The value $V_1$ is taken from the solution obtained in line 2.

For every task $T_i$, $1 \leq i \leq n-1$, we compute the interval $[\bar{t}_i^{\min}, \bar{t}_i^{\max}]$ (line 5): $\bar{t}_i^{\min}$ is the sum of execution times $\tau_k^{\min}$—given by Eq. (3) with $V^{\max}$, $M_k^{\mathrm{bc}}$, and $\overline{O}_k$—and time overheads $\delta_k$; $\bar{t}_i^{\max}$ is the sum of execution times $\tau_k^{\max}$—given by Eq. (3) with $V^{\min}$, $M_k^{\mathrm{wc}}$, and $\overline{O}_k$—and time overheads $\delta_k$.

We take then $N_i$ equally-spaced points $t_i^j$ along $[\bar{t}_i^{\min}, \bar{t}_i^{\max}]$ (line 7) and, for each such point, we compute the respective assignment $V_{i+1}^j$ (we solve the dynamic voltage scaling problem as formulated by Eqs. (19)-(23) assuming that the completion time of $T_i$ is $t_i^j$) and store it accordingly in the $j$-th position in the lookup table $\mathrm{LUT}_i$ (line 8).

**input**: The maximum number $N^{\max}$ of assignments
**output**: The set of assignments

1: distribute $N^{\max}$ among tasks ($T_i$ gets $N_i$ points)
2: solve instance $c = 0$ of the problem given by Eqs. (13)-(18); take the solution and make $\overline{O}_i := O_i$, $1 \leq i \leq n$
3: store $V_1$ in $\mathrm{LUT}_1[1]$
4: **for** $i \leftarrow 1, 2, \ldots, n-1$ **do**
5: $\quad \bar{t}_i^{\min} := \sum_{k=1}^{i} (\tau_k^{\min} + \delta_k)$; $\quad \bar{t}_i^{\max} := \sum_{k=1}^{i} (\tau_k^{\max} + \delta_k)$
6: $\quad$ **for** $j \leftarrow 1, 2, \ldots, N_i$ **do**
7: $\quad\quad t_i^j := [(N_i - j)\bar{t}_i^{\min} + j\,\bar{t}_i^{\max}]/N_i$
8: $\quad\quad$ compute $V_{i+1}^j$ for $t_i^j$ and store it in $\mathrm{LUT}_i[j]$
9: $\quad$ **end for**
10: **end for**

Algorithm 1: Off-line phase

The set of assignments, prepared off-line, is used on-line by the QS scheduler as outlined by Algorithm 2. This algorithm is called every time a task completes: upon finishing task $T_i$, the lookup table $\mathrm{LUT}_i$ is consulted. The index $j$ of the table entry is calculated very easily (line 1). Computing directly the index $j$, instead of searching through the table $\mathrm{LUT}_i$, is possible because the points $t_i^j$ stored in $\mathrm{LUT}_i$ are equally-spaced. The voltage setting stored in $\mathrm{LUT}_i[j]$ is retrieved (line 2) and finally the voltage at which task $T_{i+1}$ must run as well as the number of optional cycles it must execute is returned as a V/O assignment (line 3). Notice that Algorithm 2 has a time complexity $\mathcal{O}(1)$, which means that

the on-line operation performed by the QS scheduler takes constant time and energy. More importantly, due to the simplicity of the algorithm, this lookup and selection process is several orders of magnitude cheaper than the on-line computation by the dynamic speculative V/O scheduler.

**input**: Actual completion time $t_i$ of $T_i$ and lookup table $\text{LUT}_i$ as well as fixed optional cycles $\overline{O}_{i+1}$
**output**: The assignment $V_{i+1}/O_{i+1}$ for the next task $T_{i+1}$

1: $j := \lceil N_i(t_i - \overline{t}_i^{\min})/(\overline{t}_i^{\max} - \overline{t}_i^{\min})\rceil$
2: $V_{i+1}$ := assignment stored in $\text{LUT}_i[j]$; $O_{i+1} := \overline{O}_{i+1}$
3: return $V_{i+1}/O_{i+1}$

Algorithm 2: On-line phase

In summary, in our QS solution to the problem of minimizing energy subject to time and reward constraints, we first fix off-line the number of optional cycles assigned to each task, by taking the values $O_i$ as given by the solution to the problem formulated by Eqs. (13)-(18) (instance $c = 0$). Thus the original problem is reduced to QS voltage scaling for energy minimization. The voltage-scaling problem in a QS framework had previously been discussed by Andrei *et al.* [2]. In the one-dimension space of possible completion times, we select points and compute the corresponding voltage assignments as discussed above. For each task, a number of voltage settings are stored in its respective lookup table. Note that these tables contain only voltage values as the number of optional cycles has already been fixed off-line.

At run-time, the voltage values are simply obtained by consulting the respective lookup table each time a task completes (recall that the voltage setting read from the table depends on the completion time); this voltage value together with the number of optional cycles fixed off-line make up the V/O assignment for the next task.

## 6 Experimental Evaluation

The approach proposed in this paper has been evaluated through a large a large number of experiments using numerous synthetic benchmarks. Such synthetic examples correspond to randomly generated task graphs that contain between 10 and 100 tasks. Every point in the plots of the experimental evaluation presented in this Section (Figs. 6 through 9) corresponds to the average of the results of 75 synthetic task graphs, resulting overall in more than 2500 performed experiments. We adopted the technology-dependent parameters from [6], which 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 $\gamma_i$ chosen randomly.

The first set of experiments validates the claim that the dynamic speculative V/O scheduler (which solves the problem formulated by Eqs. (13)-(18)) outperforms the non-speculative one (which solves the problem formulated by Eqs. (9)-(12)). Fig. 6 shows the average energy savings (relative to a static V/O assignment) 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 such that the reward constraint is guaranteed). The highest savings can be obtained for systems with small deadline slack: the larger the deadline slack is, the lower the voltages given by a static assignment can be (tasks can run slower), and therefore the difference in energy consumed by a static and a dynamic solution is smaller. The experiments whose results are presented in Fig. 6 were performed considering the ideal case of zero time and energy on-line overheads and show clearly that a dynamic speculative V/O scheduler performs better (that is, produces higher energy savings) than its non-speculative counterpart.

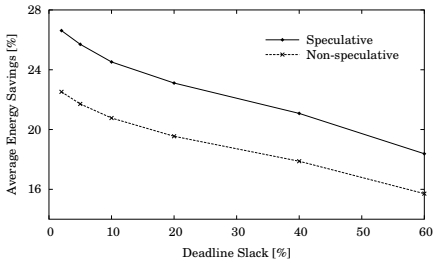
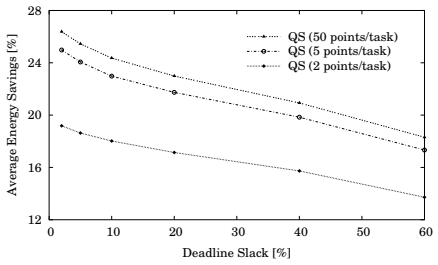



Fig. 6. Comparison of the speculative and non-speculative dynamic V/O schedulers

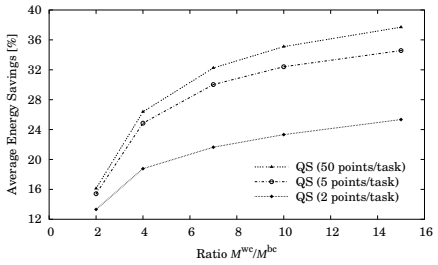
In a second set of experiments we evaluated the QS approach proposed in this paper, in terms of the energy savings achieved by it with respect to the optimal static solution. In this set of experiments we did take into consideration the time and energy overheads needed for selecting the voltage settings among the precomputed ones. In these experiments we consider that the time and energy overheads needed for selecting the assignments by the QS scheduler are $\delta^{sel} = 450$ ns and $\mathcal{E}^{sel} = 400$ 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 [7]. Fig. 7(a) shows the energy savings by our QS approach for three cases: 2, 5, and 50 points (assignments stored in the lookup tables) per task. More points per task produce naturally higher energy savings but even with a couple of points per task, as shown by the plot, very significant energy savings can be achieved (close to 20% for systems with tight deadlines).

Fig. 7(b) also shows the energy savings achieved by the QS approach, but this time as a function of the ratio between the worst-case number of cycles $M^{\text{wc}}$ and the best-case number of cycles $M^{\text{bc}}$. In these experiments we have considered systems with a deadline slack of 10%. As the ratio $M^{\text{wc}}/M^{\text{bc}}$ increases, the dynamic slack becomes larger and therefore there is more room for exploiting it in order to reduce the total energy consumed by the system.

In a third set of experiments we evaluated the quality of the solution given by the QS approach presented in this section with respect to the theoretical limit that could be achieved without knowing in advance the actual number of execution cycles (the energy consumed when a dynamic speculative V/O scheduler is used, in the ideal case of zero overheads—$\delta_i^{dyn} = 0$ and $\mathcal{E}_i^{dyn} = 0$). In order to make a fair comparison, in this set of experiments, we considered also zero overheads for the QS approach ($\delta_i^{sel} = 0$ and $\mathcal{E}_i^{sel} = 0$). Fig. 8 shows the deviation $dev = (E^{qs} - E^{ideal})/E^{ideal}$ as a function of the number of precomputed voltages (points per task), where $E^{ideal}$ is the total energy consumed for the case of an ideal dynamic speculative V/O scheduler and $E^{qs}$ is the total energy consumed for the case of a QS scheduler that selects voltages from lookup tables prepared as explained in




(a) Influence of the deadline slack




(b) Influence of the ratio $M^{\mathrm{wc}}/M^{\mathrm{bc}}$

Fig. 7. Comparison of the QS and static solutions

Section 5. In this set of experiments we have considered systems with deadline slack of 20%. It must be noted that $E^{qs}$ corresponds to the proposed QS approach in which we fix the number of optional cycles and the precomputed assignments are only voltage settings, whereas $E^{ideal}$ corresponds to the dynamic V/O scheduler that recomputes both voltage and number of optional cycles every time a task completes. Even so, with relatively few points per task it is possible to get very close to the theoretical limit, for instance, for 20 points per task the average deviation is just 0.4%.

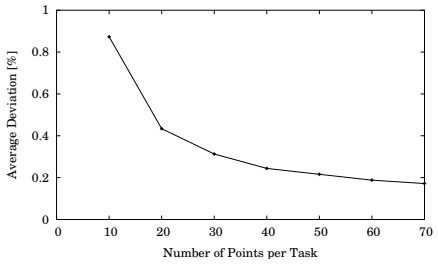



Fig. 8. Comparison of the QS and ideal dynamic solutions

Finally, in a fourth set of experiments we took into consideration realistic values for the on-line overheads $\delta_i^{dyn}$ and $\mathcal{E}_i^{dyn}$ (needed for recomputing at run-time voltage values and number of optional cycles) of the dynamic speculative V/O scheduler as well as the on-line overheads $\delta_i^{sel}$ and $\mathcal{E}_i^{sel}$ (needed for looking up the tables and selecting one of the precomputed assignments) of the QS scheduler. The overhead values used in these experiments were taken from [10], where heuristic methods were used for solving a similar problem. Fig. 9 shows the average energy savings by the dynamic and QS approaches (taking as baseline the energy consumed when using a static approach). It shows that in practice the dynamic approach makes the energy consumption higher than in the static solution (negative savings), a fact that is due to the high overheads incurred by computing on-line assignments by the dynamic V/O scheduler. Also because of the high overheads, when the system has tight deadlines, the dynamic approach cannot even guarantee the time constraints. On the contrary, the QS approach succeeds in exploiting the dynamic slack and thus reducing the energy consumption because of its low on-line overheads.

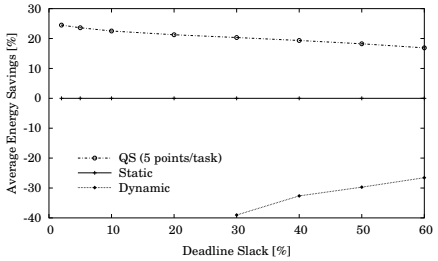



Fig. 9. Comparison considering realistic overheads

## 7 Conclusions

We have addressed the problem of minimizing the energy consumption for real-time systems with both reward and time constraints in the frame of Imprecise Computation systems. To the best of our knowledge this is the first approach presented for this particular problem.

The proposed approach has as chief merit the ability to effectively exploit the dynamic slack, caused by tasks executing less clock cycles than in the worst case. Such a QS approach succeeds in exploiting the dynamic slack, yet incurring a very low on-line overhead, because the complex time- and energy-consuming parts of the computations are performed off-line, at design-time, leaving for run-time only simple lookup and selection operations.

The evaluation of our solution has been performed using a large number of synthetic benchmarks. These have shown that significant reductions in the energy consumption can be achieved with our technique, for instance, energy savings of around 20% for systems with tight deadlines.

## 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*, pp. 518–523, 2004.

[2] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints. In *Proc. DATE*, pp. 514–519, 2005.

[3] L. A. Cortés. *Verification and Scheduling Techniques for Real-Time Embedded Systems*. PhD thesis, Dept. Computer and Information Science, Linköping University, Mar. 2005.

[4] 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*, pp. 889–894, 2005.

[5] J. W. S. Liu, W.-K. Shih, K.-J. Lin, and R. Bettati. Imprecise Computations. *Proc. IEEE*, 82(1):83–94, Jan. 1994.

[6] 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*, pp. 721–725, 2002.

[7] NEC Memories. `http://www.necel.com/memory/index_e.html`.

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

[9] T. Okuma, H. Yasuura, and T. Ishihara. Software Energy Reduction Techniques for Variable-Voltage Processors. *IEEE Design & Test of Computers*, 18(2):31–41, Mar. 2001.

[10] C. Rusu, R. Melhem, and D. Mossé. Maximizing Rewards for Real-Time Applications with Energy Constraints. *ACM Trans. on Embedded Computing Systems*, 2(4):537–559, Nov. 2003.

[11] S. A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, New York, NY, 1991.