Quasi-Static Assignment of Voltages and Optional Cycles for Maximizing Rewards in Real-Time Systems with Energy Constraints

Luis Alejandro Cortés [1,2]
alejandro.cortes@volvo.com

Petru Eles [2]
petel@ida.liu.se

Zebo Peng [2]
zebpe@ida.liu.se

[1] Volvo Truck Corporation
Gothenburg, Sweden

[2] Linköping University
Linköping, Sweden

ABSTRACT

There exist real-time systems for which it is possible to trade off precision for timeliness. In these cases, a function assigns reward to the application depending on the amount of computation allotted to it. At the same time, many such applications run on battery-powered devices with stringent energy constraints. This paper addresses the problem of maximizing rewards subject to time and energy constraints. We propose a quasi-static approach where the problem is solved in two steps: first, at design-time, a number of solutions are computed and stored (off-line phase); second, one of the precomputed solutions is selected at run-time based on actual values of time and energy (on-line phase). Thus our approach is able to exploit, with low on-line overhead, the dynamic slack caused by tasks executing less number of cycles than in the worst case. We conduct numerous experiments in order to show the advantages of our approach.

Categories & Subject Descriptors: C.3 [Special-Purpose and Application-Based Systems]: Real-Time and Embedded.
General Terms: Algorithms, Design.
Keywords: Quasi-Static, Dynamic Voltage Scaling.

1. INTRODUCTION

The trade-off between energy consumption and performance has extensively been studied under the framework of Dynamic Voltage Scaling (DVS) [8], [5].

There exist real-time applications, such as image processing and multimedia, in which approximate but timely results are acceptable. Fuzzy images in time are often preferable to perfect images too late. Thus it is possible to trade off precision for timeliness. Such systems have been studied under the Imprecise Computation (IC) model [4], where 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. A function associated with each task assigns reward as a function of the amount of computation allotted to it: the more the optional part executes, the more reward it produces.

While DVS techniques have mostly been studied in the context of hard real-time systems, IC approaches have until now disregarded the power/energy aspects. Rusu *et al.* [9] proposed recently the first approach in which energy, reward, and deadlines are considered under a unified framework. Their goal is to maximize the total reward without exceeding deadlines or the available energy. This approach solves the optimization problem statically, at compile-time, and therefore considers only worst cases. Such an approach can only exploit the *static* slack, which is due to the fact that at nominal voltage the processor runs faster than needed.

Dynamic approaches have been used for hard real-time systems in order to exploit the *dynamic* slack, which is caused by tasks executing less cycles than in the worst case. In this paper we consider tasks composed of mandatory and optional parts and we aim at finding a Voltage/Optional-cycles (V/O) assignment (actually a set of assignments as explained later), such that the total reward is maximal while guaranteeing the deadlines and the energy budget. We exploit the dynamic time and energy slack caused by variations in the actual number of execution cycles. Furthermore, we consider the time and energy overhead incurred during voltage transitions.

Static V/O assignment refers to finding at design-time one assignment of voltages and optional cycles that makes the reward maximal while guaranteeing the time and the energy constraints (this is the problem addressed by [9]). *Dynamic* V/O assignment refers to finding at run-time, every time a task completes, a new V/O assignment for tasks not yet started, but considering the actual execution times and energy values. Static V/O assignment causes no on-line overhead, but it is pessimistic because actual execution times are typically far off from worst-case values. Dynamic V/O assignment exploits information known only after tasks complete and accordingly computes new assignments aiming at improving the reward, but the energy and time overhead for on-line computations is high, even if polynomial-time algorithms can be used. We propose a *Quasi-Static (QS)* approach that is able to exploit, with low on-line overhead, the dynamic slack: first, at design-time, a set of V/O assignments are computed and stored (off-line phase); second, the selection among the precomputed assignments is left for run-time, based on actual completion times and consumed energy (on-line phase).

QS scheduling for maximizing utility in hard/soft real-time systems was recently discussed [1], but without any energy consideration. To our knowledge, this is the first work that considers reward, energy, and deadlines in a QS framework.

2. PRELIMINARIES

2.1 Task and Architectural Models

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 tasks and the edges $\mathbf{E}$ indicate the data dependencies between tasks. Each task T_i has a mandatory 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^{bc} and the worst-case number of cycles M_i^{wc} ($M_i^{\mathrm{bc}} \leq M_i \leq M_i^{\mathrm{wc}}$). 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 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

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

to T_i. We consider non-decreasing concave reward functions, as they capture the particularities of most real-life applications [9]. We assume 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 denoted $R = \sum_{T_i \in \mathbf{T}} R_i(O_i)$.

We consider that tasks are non-preemtable 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 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 [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 $[V^{\min}, V^{\max}]$.

In our QS approach we compute a number of V/O assignments. This set of assignments is stored in a dedicated memory in the form of lookup tables, one table LUT_i for each task T_i. The maximum number of V/O assignments that can be stored is a parameter $N^{\max}$ fixed by the designer.

2.2 Energy and Delay Models

For the sake of clarity, we consider only the dynamic energy consumption. Nonetheless, the leakage energy as well as Adaptive Body Biasing (ABB) techniques [5] can easily be incorporated into the formulation without changing our general approach. The dynamic energy consumed by task T_i is given by [5]

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

where C_i is the effective switched capacitance, V_i is the supply voltage, and $M_i + O_i$ is the number of cycles executed by T_i. The energy overhead, for switching from V_i to V_j, is [5]

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

where C_r is the capacitance of the power rail. We also consider, for the QS solution, the energy overhead $\mathcal{E}_i^{sel}$ caused by looking up and selecting one of the precomputed assignments. The way we store the assignments makes the selection process take $\mathcal{O}(1)$ time and thus $\mathcal{E}_i^{sel}$ is a constant value. 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, that is, $\mathcal{E}_i^{dyn} = \mathcal{E}_i^{sel}$.

The total energy consumed up to the completion of task T_i is denoted EC_i. We consider a given energy budget $E^{\max}$ that imposes a constraint on the total amount of energy.

The execution time of a task T_i executing $M_i + O_i$ cycles at V_i is given by [5]

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

where k is a technology-dependent constant, α is the saturation velocity index ($1.4 \leq \alpha \leq 2$), and V_{th} is the threshold voltage. The time overhead, for switching from V_i to V_j, is given by [5]

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

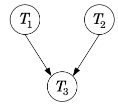
where p is a constant. The time overhead for looking up and selecting one V/O assignment in the QS approach is denoted δ_i^{sel} and, as explained above, is constant.

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 δ_i captures the different time overheads. $\delta_i = \delta_{i-1,i}^{\Delta V} + \delta_i^{dyn}$ where δ_i^{dyn} is the on-line overhead. This on-line overhead in a QS solution is just the lookup and selection time, that is, $\delta_i^{dyn} = \delta_i^{sel}$.

3. MOTIVATIONAL EXAMPLE

Let us consider the example shown in Fig. 1. The reward functions are of the form $R_i(O_i) = K_i O_i,\ O_i \leq O_i^{\max}$. The energy budget is $E^{\max} = 1$ mJ and the tasks run on a processor with continuous voltage scaling in the range 0.6-1.8 V. In this example we assume that transition overheads are zero.

The optimal static V/O assignment is given by Table 1(a). It 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.



Task	M_i^{bc}	M_i^{wc}	C_i [nF]	d_i [μs]	K_i	$O_i^{\max}$
T_1	20000	100000	0.7	250	0.00014	50000
T_2	70000	160000	1.2	600	0.0002	80000
T_3	100000	180000	0.9	1000	0.0001	60000

Fig. 1: Motivational example

This assignment produces a total reward $R^{st} = 3.99$.

The actual number of execution cycles, which are not known in advance, are typically far off from the worst-case values used to compute the static V/O assignment. The assignment could instead be computed dynamically and thus exploit the dynamic slack: taking into account the information about completion time and consumed energy, a new V/O assignment is computed every time a task finishes. For instance, for the situation $M_1 = 60000$, $M_2 = 100000$, $M_3 = 150000$, the dynamic V/O assignment in the ideal case (on-line computations take zero time and energy) is given by Table 1(b). This assignment delivers a total reward $R^{dyn^{ideal}} = 16.28$. In reality, however, the on-line overhead caused by computing new assignments is not negligible. When considering time and energy overheads, using for example $\delta^{dyn} = 65$ μs and $\mathcal{E}^{dyn} = 55$ μJ, the assignment computed dynamically is clearly different, as given by Table 1(c). This assignment yields a total reward $R^{dyn^{real}} = 6.26$. The values of δ^{dyn} and $\mathcal{E}^{dyn}$ are in practice orders of magnitude higher than the ones used in this hypothetical example [2]. Even on-line heuristics, which produce approximate results, have long execution times [9].

In our QS approach we compute at design-time a number of assignments, which are selected at run-time by the so-called quasi-static V/O scheduler. We can define, for instance, a set of assignments as given by Fig. 2. When finishing each task, V_i and O_i are selected from the precomputed set, according to the given condition. These assignments were computed considering selection overheads $\delta^{sel} = 0.3$ μs and $\mathcal{E}^{sel} = 0.3$ μJ.

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

Fig. 2: Precomputed set of V/O assignments

For $M_1 = 60000$, $M_2 = 100000$, $M_3 = 150000$, and the set in Fig. 2, 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, $t_1 = \tau_1 = 111.73 \leq 130$ μs and $EC_1 = E_1 = 114.97 \leq 135$ μJ, $V_2 = 1.446/O_2 = 43446$ is selected. Task T_2 runs under this assignment so that, when it finishes, $t_2 = \tau_1 + \delta_2^{sel} + \tau_2 = 442.99$ μs and $EC_2 = E_1 + \mathcal{E}_2^{sel} + E_2 = 474.89$ μJ. Then $V_3 = 1.486/O_3 = 46473$ is selected and task T_3 is executed accordingly. Table 1(d) summarizes the selected assignment, which delivers a total reward $R^{qs} = 13.34$ (compare to $R^{dyn^{ideal}} = 16.28$, $R^{dyn^{real}} = 6.26$, and $R^{st} = 3.99$).

4. PROBLEM FORMULATION

In what follows we present the precise formulation of related problems and the particular problem addressed in this paper.

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

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

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

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

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

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

(a) Static

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

(b) Dynamic ($\delta^{dyn}\!=\!0$, $\mathcal{E}^{dyn}\!=\!0$)

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

(c) Dynamic ($\delta^{dyn}\!=\!65\mu$s, $\mathcal{E}^{dyn}\!=\!55\mu$J)

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

(d) QS selected from Fig. 2

Table 1: V/O assignments (for $M_1\!=\!60000$, $M_2\!=\!100000$, $M_3\!=\!150000$)

The total reward, as given by Eq. (5), is to be maximized. For each task the voltage V_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 time τ_i, and tasks must complete before their deadlines (Eq. (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 energy budget $E^{\max}$ (Eq. (8)). Note that a static assignment must consider the worst-case number of mandatory cycles M_i^{wc} for every task (Eqs. (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, we obtain a solution that is very near to the optimal one [2]. We can rewrite the above problem as "minimize $\sum R'_i(O_i)$", with $R'_i(O_i)\!=\!-R_i(O_i)$. It can thus be noted that: $R'_i(O_i)$ is a convex function since $R_i(O_i)$ is concave (see Subsection 2.1); the constraint functions are also convex. Therefore it corresponds to a *convex non-linear programming* (NLP) formulation [7]. It is worth mentioning that convex NLP problems can be solved using polynomial-time methods [7].

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 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

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

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

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

$$\sum_{i=c+1}^{n} (\mathcal{E}_i^{dyn} + \mathcal{E}^{\Delta V}_{i-1,i} + E_i) \le E^{\max} - EC_c$$

where δ_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.

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 EC_c), but considering δ_i^{dyn} and $\mathcal{E}_i^{dyn}$. The total reward R^{ideal} delivered by a dynamic V/O scheduler in the ideal case $\delta_i^{dyn}\!=\!0$, $\mathcal{E}_i^{dyn}\!=\!0$ 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.

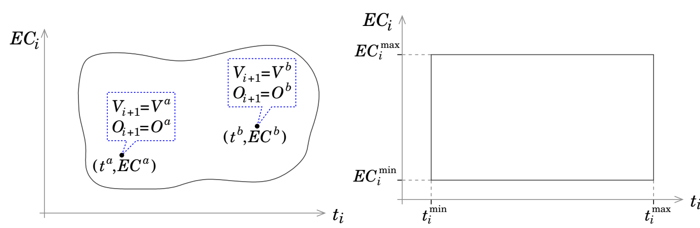
We prepare off-line a set of V/O assignments, one of which is to be selected by the *quasi-static V/O scheduler*. When a task T_c completes, the quasi-static V/O scheduler checks the completion time t_c and the total energy EC_c, and looks up an assignment in the table for T_c. From the lookup table LUT$_c$, it obtains the point (t'_c, EC'_c)—the closest to (t_c, EC_c) such that $t_c \le t'_c$ and $EC_c \le EC'_c$—and selects V'/O' corresponding to (t'_c, 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 R^{qs}, as delivered by the quasi-static V/O scheduler, as high as possible. The problem we discuss in the rest of the paper is the following:

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

As discussed in Section 5, for a task T_i, potentially there exist infinitely many values for t_i and EC_i. Therefore, in order to approach the theoretical limit R^{ideal}, it would be needed to compute an infinite number of V/O assignments, one for each (t_i, 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^{ideal}.

5. SET OF V/O ASSIGNMENTS

For each task T_i, there exists a space time-energy of possible values of completion time t_i and energy EC_i consumed up to the completion of T_i (see Fig. 3(a)). 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 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 $t_i = t^a$, $EC_i = EC^a$. Note that different points (t_i, EC_i) define different assignments.



(a) General view (b) Pessimistic boundaries

Fig. 3: Space time-energy

We need first to determine the boundaries of the space time-energy 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 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^{bc}, M_j^{wc}, and $O_j^{\max}$, $1 \le j \le i$. 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^{bc} and zero optional cycles at maximum voltage $V^{\max}$, while $t_i^{\max}$ occurs when each task T_j executes $M_j^{\mathrm{wc}} + O_j^{\max}$ cycles at $V^{\min}$. Similarly, $EC_i^{\min}$ happens when each task T_j executes M_j^{bc} cycles at $V^{\min}$, while $EC_i^{\max}$ happens when each task T_j executes $M_j^{\mathrm{wc}} + O_j^{\max}$ cycles at $V^{\max}$. The intervals $[t_i^{\min}, t_i^{\max}]$ and $[EC_i^{\min}, EC_i^{\max}]$ bound the space time-energy as shown in Fig. 3(b). However, the space time-energy delimited in this way is rather pessimistic as there are points in this space that cannot happen. For instance, $(t_i^{\min}, EC_i^{\min})$ is not feasible because $t_i^{\min}$ requires all tasks running at $V^{\max}$ whereas $EC_i^{\min}$ requires all tasks running at $V^{\min}$.

5.1 Characterization of the Space Time-Energy

We take now a closer look at the relation between the energy E_i consumed by a task and its execution time τ_i as given by

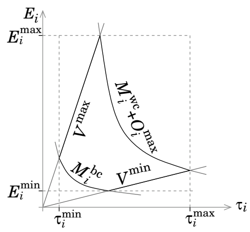


Fig. 4: Space τ_i-E_i for task T_i

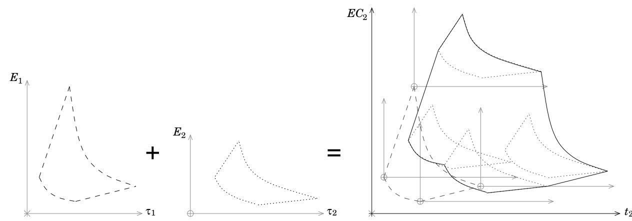


Fig. 5: "Sum" of spaces τ_1-E_1 and τ_2-E_2

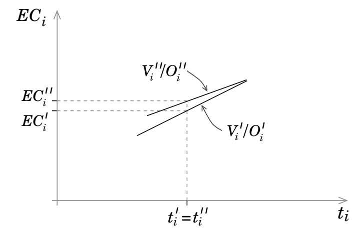


Fig. 6: V_i'/O_i' and V_i''/O_i'' converge

Eqs. (1) and (3). In this subsection we consider, as commonly assumed in the literature [8], that τ_i is inversely proportional to V_i ($V_{th} = 0$, $\alpha = 2$) to make the illustration of our point simpler, yet the drawn conclusions are valid in general. After simple algebraic manipulations on Eqs. (1) and (3) we get

$$E_i = \frac{C_i V_i^3}{k} \tau_i \tag{9}$$

which, in the space τ_i-E_i, gives a family of straight lines, each for a particular V_i. Thus $E_i = C_i (V^{\min})^3 \tau_i / k$ and $E_i = C_i (V^{\max})^3 \tau_i / k$ define two boundaries in the space τ_i-E_i. We can also write

$$E_i = C_i k^2 (M_i + O_i)^3 \frac{1}{\tau_i^2} \tag{10}$$

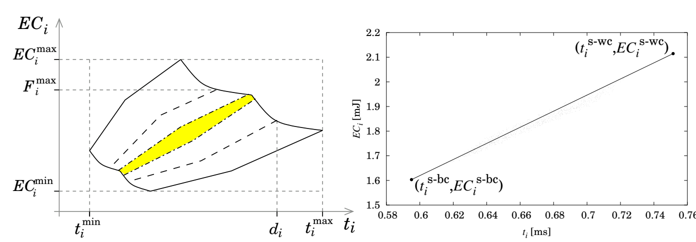
which gives a family of curves, each for a particular $M_i + O_i$. Thus $E_i = C_i k^2 (M_i^{\mathrm{bc}})^3 / \tau_i^2$ and $E_i = C_i k^2 (M_i^{\mathrm{wc}} + O_i^{\max})^3 / \tau_i^2$ define another two boundaries, as shown in Fig. 4. Note that Fig. 4 represents the energy consumed by *one* task (energy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

In order to obtain a realistic view of the diagram in Fig. 3(b), we must "sum" the spaces τ_j-E_j introduced above. The result of this summation, as illustrated in Fig. 5, gives the space time-energy t_i-EC_i we are interested in. In Fig. 5 the space t_2-EC_2 is obtained by sliding the space τ_2-E_2 with its coordinate origin along the boundaries of τ_1-E_1.

The shape of the space t_i-EC_i is depicted by the solid lines in Fig. 7(a). There are in addition deadlines d_i to consider as well as energy constraints $F_i^{\max}$. Note that, for each task, the deadline d_i is explicitly given as part of the system model whereas $F_i^{\max}$ is an implicit constraint induced by the overall energy constraint $E^{\max}$. The constraint $F_i^{\max}$ comes from the fact that future tasks will consume at least a certain amount of energy $F_{i+1\to n}$ so that $F_i^{\max} = E^{\max} - F_{i+1\to n}$. The deadline d_i and the induced energy constraint $F_i^{\max}$ further restrict the space time-energy, as depicted by the dashed lines in Fig. 7(a).

The space time-energy 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, as 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 consumed energy 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 energy EC''_{i-1}, T_i runs under the assignment V_i''/O_i''. Since the points (t'_{i-1}, EC'_{i-1}) and (t''_{i-1}, 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 space t_i-EC_i 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 $M_i^{\mathrm{bc}} + O_i'$ cycles at V_i' and the other end corresponding to the execution of $M_i^{\mathrm{wc}} + O_i'$ cycles at V_i' [2]. V_i''/O_i'' defines analogously a straight line L_i''. Solutions to the dynamic V/O assignment problem, though, attempt to make 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 lines L_i' and L_i'') will thus converge in the space t_i-EC_i when $M_i' = M_i'' = M_i^{\mathrm{wc}}$ (which is the value that has to be assumed when computing the solutions) as shown in Fig. 6. 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 EC_i' and EC_i'' will actually be very close (see Fig. 6). This means that in practice the space t_i-EC_i is a narrow area, as depicted by the dash-dot lines and the gray area enclosed by them in Fig. 7(a).

We conducted a number of experiments in order to determine how narrow the area in the space time-energy actually is. For each task T_i, we considered a segment of straight line, called in the sequel the *diagonal* D_i, defined by the points $(t_i^{\text{s-bc}}, EC_i^{\text{s-bc}})$ and $(t_i^{\text{s-wc}}, EC_i^{\text{s-wc}})$. The point $(t_i^{\text{s-bc}}, EC_i^{\text{s-bc}})$ 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^{bc}. Analogously, $(t_i^{\text{s-wc}}, EC_i^{\text{s-wc}})$ corresponds to the solution in the particular case when every task T_j executes its worst-case number of mandatory cycles M_j^{wc}. We generated 50 synthetic examples, consisting of between 10 and 100 tasks, and we simulated for each of them the ideal dynamic V/O scheduler for 1000 cases, each case S being a combination of executed mandatory cycles $M_1^S, M_2^S, \ldots, M_n^S$. 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, EC_i^S) in the space t_i-EC_i, as given by the ideal dynamic V/O scheduler. Each point (t_i^S, EC_i^S) was compared with the point $(t_i^S, EC_i^{D_i})$ (a point with the same abscissa t_i^S, but on the diagonal D_i so that its ordinate is $EC_i^{D_i}$) and the relative deviation $e = |EC_i^S - EC_i^{D_i}|/EC_i^S$ was computed. From the simulations we found average deviations of around 1% and a maximum deviation of 4.5%. These results show that the space t_i-EC_i is indeed a narrow area. Fig. 7(b) shows the actual points (t_i^S, EC_i^S), corresponding to the simulation of the 1000 sets S of executed mandatory cycles, in the space time-energy of a particular task T_i as well as the diagonal D_i.



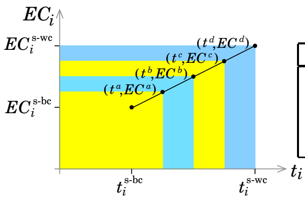
(a) Realistic boundaries (b) Actual points

Fig. 7: Realistic view of the space time-energy

5.2 Point Selection and Assignment Computation

We conclude, from the discussion in Subsection 5.1, that the points in the space t_i-EC_i are concentrated in a narrow area along the diagonal D_i. This observation is crucial for choosing the points for which we compute the V/O assignments.

We take N_i points (t_i^j, EC_i^j), $1 \leq j \leq N_i$, along the diagonal D_i in the space t_i-EC_i 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 EC_i^j. For the computation of the assignment V_{i+1}^j/O_{i+1}^j, the time and energy overheads δ_{i+1}^{sel} and $\mathcal{E}_{i+1}^{sel}$ (needed for selecting assignments at run-time) are taken into account. Each of the chosen points together with its respective V/O assignment covers a region as indicated in Fig. 8.



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

Fig. 8: Regions

The pseudocode of the procedure for computing the set of assignments is given by Alg. 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 6, it is more convenient to distribute the assignments according to the size of the space time-energy of tasks (we use the length of the diagonal D as a measure of this size), in such a way that lookup tables of tasks with larger spaces get more points.

After distributing $N^{\max}$ among tasks, the solutions V/O$^{\text{s-bc}}$ and V/O$^{\text{s-wc}}$ are computed (lines 2-3). V/O$^{\text{s-bc}}$ (V/O$^{\text{s-wc}}$) is a structure that contains the pairs $V_i^{\text{s-bc}}/O_i^{\text{s-bc}}$ ($V_i^{\text{s-wc}}/O_i^{\text{s-wc}}$), $1 \le i \le n$, as computed by the dynamic V/O scheduler when every task executes its best-case (worst-case) number of cycles. Since the assignment V_1/O_1 is invariably the same, 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: a) $t_i^{\text{s-bc}}$ ($t_i^{\text{s-wc}}$) as the sum of execution times $\tau_k^{\text{s-bc}}$ ($\tau_k^{\text{s-wc}}$)—given by Eq. (3) with $V_k^{\text{s-bc}}$, M_k^{bc}, and $O_k^{\text{s-bc}}$ ($V_k^{\text{s-wc}}$, M_k^{wc}, and $O_k^{\text{s-wc}}$)—and time overheads δ_k (line 7); b) $EC_i^{\text{s-bc}}$ ($EC_i^{\text{s-wc}}$) as the sum of energies $E_k^{\text{s-bc}}$ ($E_k^{\text{s-wc}}$)—given by Eq. (1) with $V_k^{\text{s-bc}}$, M_k^{bc}, and $O_k^{\text{s-bc}}$ ($V_k^{\text{s-wc}}$, M_k^{wc}, and $O_k^{\text{s-wc}}$)—and energy overheads $\mathcal{E}_k$ (line 8). For every task T_i, we take N_i equally-spaced points (t_i^j, EC_i^j) along the diagonal D_i (straight line segment from $(t_i^{\text{s-bc}}, EC_i^{\text{s-bc}})$ to $(t_i^{\text{s-wc}}, EC_i^{\text{s-wc}})$) and, for each such point, we compute the respective assignment V_{i+1}^j/O_{i+1}^j and store it in the j-th position in the particular lookup table LUT$_i$ (lines 10-12).

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: V/O$^{\text{s-bc}}$:= sol. by dyn. scheduler when $M_k = M_k^{\text{bc}}$, $1 \le k \le n$
3: V/O$^{\text{s-wc}}$:= sol. by dyn. scheduler when $M_k = M_k^{\text{wc}}$, $1 \le k \le n$
4: $V_1 := V_1^{\text{s-bc}} = V_1^{\text{s-wc}}$; $O_1 := O_1^{\text{s-bc}} = O_1^{\text{s-wc}}$
5: store V_1/O_1 in LUT$_1$[1]
6: **for** $i \leftarrow 1, 2, \ldots, n-1$ **do**
7: $\quad t_i^{\text{s-bc}} := \sum_{k=1}^{i} (\tau_k^{\text{s-bc}} + \delta_k)$; $\quad t_i^{\text{s-wc}} := \sum_{k=1}^{i} (\tau_k^{\text{s-wc}} + \delta_k)$
8: $\quad EC_i^{\text{s-bc}} := \sum_{k=1}^{i} (E_k^{\text{s-bc}} + \mathcal{E}_k)$; $\quad EC_i^{\text{s-wc}} := \sum_{k=1}^{i} (E_k^{\text{s-wc}} + \mathcal{E}_k)$
9: $\quad$ **for** $j \leftarrow 1, 2, \ldots, N_i$ **do**
10: $\quad\quad t_i^j := [(N_i - j) t_i^{\text{s-bc}} + j\, t_i^{\text{s-wc}}]/N_i$
11: $\quad\quad EC_i^j := [(N_i - j) EC_i^{\text{s-bc}} + j\, EC_i^{\text{s-wc}}]/N_i$
12: $\quad\quad$ compute V_{i+1}^j/O_{i+1}^j for (t_i^j, EC_i^j) and store it in LUT$_i[j]$
13: $\quad$ **end for**
14: **end for**

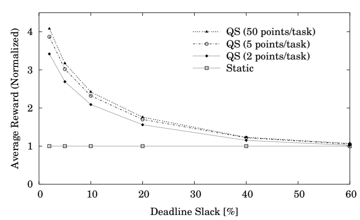
Algorithm 1: OffLinePhase

At run-time, the selection of assignments by the quasi-static V/O scheduler is very simple: upon completing task T_i, the lookup table LUT$_i$ is consulted and the index j of the table entry is calculated directly (without searching through the table LUT$_i$). Then the V/O assignment in LUT$_i[j]$ is retrieved. The on-line operation performed by the quasi-static V/O scheduler takes takes constant time and energy and it is several orders of magnitude cheaper than the on-line computation by the dynamic V/O scheduler.

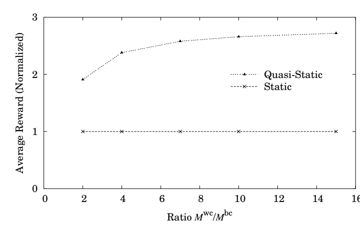
6. EXPERIMENTAL RESULTS

We evaluated our approach through numerous synthetic benchmarks. We considered task graphs containing between 10 and 100 tasks. Each point in the plots of the experimental results (Figs. 9, 10, and 11) corresponds to 50 automatically-generated task graphs. The technology-dependent parameters were adopted from [5] and correspond to a processor in a 0.18 μm CMOS fabrication process.

The first set of experiments was performed to investigate the reward gain achieved by our approach compared to the optimal static solution (the approach proposed in [9]). In these experiments we consider that the selection overheads by the quasi-static V/O scheduler are δ^{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 of, for example, a low-power Static RAM are around 70 ns and 20 nJ respectively [6]. Fig. 9(a) shows the reward (normalized with respect to the 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 QS approach (2, 5, and 50 points per task) are considered. Very significant gains in terms of total reward, up to four times, can be obtained with the QS solution, even with few points per task. The highest reward gains are achieved when the system has very tight deadlines (small slack): when large amounts of slack are available, the static solution can accommodate most of the optional cycles (there is a value $O_i^{\max}$ after which no extra reward is achieved).



(a) Influence of the deadline slack



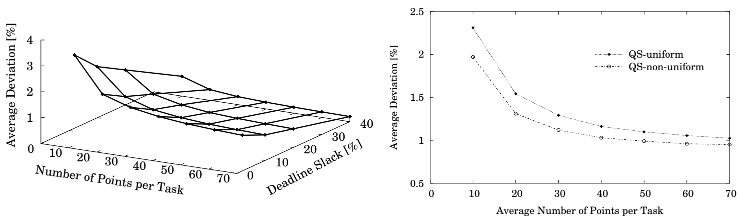
(b) Influence of ratio $M^{\text{wc}}/M^{\text{bc}}$

Fig. 9: Comparison of the quasi-static and static solutions

The influence of the ratio between the worst-case number of cycles M^{wc} and the best-case number of cycles M^{bc} has also been studied and the results are presented in Fig. 9(b). In this case we have considered systems with a deadline slack of 10% and 20 points per task in the QS solution. The larger the ratio $M^{\text{wc}}/M^{\text{bc}}$ is, the more the actual number of execution cycles deviate from the worst-case value M^{wc} (which is the value 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 QS 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). For comparison fairness, we considered zero time and energy overheads δ^{sel} and $\mathcal{E}^{sel}$. Fig. 10(a) shows the deviation $dev = (R^{ideal} - R^{qs})/R^{ideal}$ as a function of the number of precomputed assignments (points per task) and for various degrees of deadline tightness. More points per task produce higher reward, closer to the theoretical limit (smaller deviation). Nonetheless, with relatively few points per task we can get very close to the theoretical limit, for instance, for deadline slack of 20% and 30 points per task the average deviation is around 1.3%. The deviation gets smaller as the deadline slack increases, as shown in Fig. 10(a).

In the previous experiments we 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 spaces time-energy (more assignments in the lookup tables of tasks that have larger spaces), better results are obtained as shown in Fig. 10(b). This figure plots the cases of equal-size lookup tables (QS-uniform) and assignments distributed non-uniformly among tables (QS-non-uniform), as described above, for systems with a deadline slack of 20%. The abscissa is the *average* number of points per task.



(a) Influence of the deadline slack and number of points

(b) Influence of the distribution of points among lookup tables

Fig. 10: Comparison of quasi-static and ideal dyn. solutions

In a third set of experiments we took into account the on-line overheads of the dynamic V/O scheduler (as well as the QS one) and compared the static, QS, and dynamic approaches in the same graph. Fig. 11 shows the reward normalized with respect to the one by the static solution. It shows that, in a realistic setting, the dynamic approach performs poorly, even worse than the static one. Moreover, for systems with tight deadlines, 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%). The overhead values considered for the dynamic case correspond actually to overheads by heuristics [9] and not by exact methods, although in the experiments the exact solutions were considered. This means that, even in the optimistic case of an on-line algorithm that delivers exact solutions in a time frame similar to the one of heuristic methods, the QS approach outperforms the dynamic one.

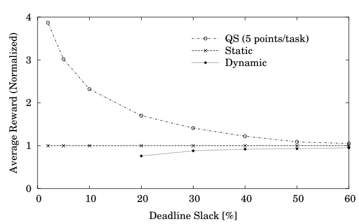


Fig. 11: Comparison considering realistic overheads

We evaluated also our approach by means of a real-life application, namely the navigation controller of an autonomous rover for exploring a remote place [3]. 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 towards its destination avoiding nearby obstacles. This application includes several 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 and estimates 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. 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 [3]. Rewards produced by 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, which in turn implies a more accurate estimation of the location and thus makes the rover get faster to its destination (that is, higher total reward). Other tasks make likewise their individual contribution to the global reward.

The navigation controller is activated periodically every 360 ms and tasks have a deadline equal to the period. 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 daytime. We use two memories, one for the assignments used during daytime and one for the set used during the night, and assume that $N^{\max} = 512$ assignments can be stored in each memory. We computed, for both cases, the sets of assignments using Alg. 1. When compared to the respective static solutions, our QS 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.

7. CONCLUSIONS

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

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 voltage levels that replace the calculated ideal one [8].

We evaluated our approach through numerous synthetic benchmarks and a realistic application. We found that significant gains in terms of reward can be obtained by the QS approach. We showed also that, due to its large on-line overheads, a dynamic approach performs poorly. Thus, the dynamic slack can efficiently be exploited only if high overheads are avoided, as done by our QS approach.

8. REFERENCES

[1] L. A. Cortés, P. Eles, and Z. Peng. Quasi-Static Scheduling for Real-Time Systems with Hard and Soft Tasks. In *Proc. DATE Conference*, pp. 1176–1181, 2004.

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

[3] D. L. Hull. *An Environment for Imprecise Computations*. PhD thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, Jan. 2000.

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

[5] 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.

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

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

[8] 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.

[9] 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.