

Energy-aware scheduling under reliability and makespan constraints

Guillaume Aupy* Anne Benoit*

Yves Robert*[†]

{guillaume.aupy, anne.benoit, yves.robert}@ens-lyon.fr

November 21, 2018

Abstract

We consider a task graph mapped on a set of homogeneous processors. We aim at minimizing the energy consumption while enforcing two constraints: a prescribed bound on the execution time (or makespan), and a reliability threshold. Dynamic voltage and frequency scaling (DVFS) is an approach frequently used to reduce the energy consumption of a schedule, but slowing down the execution of a task to save energy is decreasing the reliability of the execution. In this work, to improve the reliability of a schedule while reducing the energy consumption, we allow for the re-execution of some tasks. We assess the complexity of the tri-criteria scheduling problem (makespan, reliability, energy) of deciding which task to re-execute, and at which speed each execution of a task should be done, with two different speed models: either processors can have arbitrary speeds (CONTINUOUS model), or a processor can run at a finite number of different speeds and change its speed during a computation (VDD-HOPPING model). We propose several novel tri-criteria scheduling heuristics under the continuous speed model, and we evaluate them through a set of simulations. The two best heuristics turn out to be very efficient and complementary.

*Ecole Normale Supérieure de Lyon, France

[†]University of Tennessee Knoxville, USA

1 Introduction

Energy-aware scheduling has proven an important issue in the past decade, both for economical and environmental reasons. This holds true for traditional computer systems, not even to speak of battery-powered systems. More precisely, a processor running at speed s dissipates s^3 watts per unit of time [4, 6, 8], hence it consumes $s^3 \times d$ joules when operated during d units of time. To help reduce energy dissipation, processors can run at different speeds. A widely used technique to reduce energy consumption is *dynamic voltage and frequency scaling (DVFS)*, also known as speed scaling [4, 6, 8]. Indeed, by lowering supply voltage, hence processor clock frequency, it is possible to achieve important reductions in power consumption; faster speeds allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. There are two popular models for processor speeds. In the CONTINUOUS model, processors can have arbitrary speeds, and can vary them continuously in the interval $[f_{\min}, f_{\max}]$. This model is unrealistic (any possible value of the speed, say $\sqrt{e^\pi}$, cannot be obtained), but it is theoretically appealing [6]. In the VDD-HOPPING model, a processor can run at a finite number of different speeds (f_1, \dots, f_m) . It can also change its speed during a computation (*hopping* between different voltages, and hence speeds). Any rational speed can therefore be simulated [15]. The energy consumed during the execution of one task is the sum, on each time interval with constant speed f , of the energy consumed during this interval at speed f .

Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application. Obviously, this goal makes sense only when coupled with some performance bound to achieve, otherwise, the optimal solution always is to run each processor at the slowest possible speed. In this paper, we consider a directed acyclic graph (DAG) of n tasks with precedence constraints, and the goal is to schedule such an application onto a fully homogeneous platform consisting of p identical processors. This problem has been widely studied with the objective of minimizing the total execution time, or *makespan*, and it is well known to be NP-complete [7]. Since the introduction of DVFS, many papers have dealt with the optimization of energy consumption while enforcing a deadline, i.e., a bound on the makespan [4, 6, 8, 3].

There are many situations in which the mapping of the task graph is given, say by an ordered list of tasks to execute on each processor, and we do not have the freedom to change the assignment of a given task. Such a problem occurs when optimizing for legacy applications, or accounting for affinities between tasks and resources, or even when tasks are pre-allocated [19], for example for security reasons. While it is not possible to change the allocation of a task, it is possible to change its speed. This technique, which consists in exploiting the slack due to workload variations, is called *slack reclaiming* [13, 18]. In our previous work [3], assuming that the mapping and a deadline are given, we have assessed the impact of several speed variation models on the complexity of the problem of minimizing the energy consumption. Rather than using a local approach such as backfilling [22, 18], which only reclaims gaps in the schedule, we have considered the problem as a whole.

While energy consumption can be reduced by using speed scaling techniques, it was shown in [25, 10] that reducing the speed of a processor increases the number of transient fault rates of the system; the probability of failures increases exponentially, and this probability cannot be neglected in large-scale computing [16]. In order to make up for the loss in *reliability* due to the energy efficiency, different models have been proposed for fault-tolerance: (i) *re-execution* is the model under study in this work, and it consists in re-executing a task that does not meet the reliability constraint; it was also studied in [25, 24, 17]; (ii) *replication* was studied in [1, 12]; this model consists in executing the same task on several processors simultaneously, in order to meet the reliability constraints; and (iii) *checkpointing* consists in "saving" the work done at some certain points of the work, hence reducing the amount of work lost when a failure occurs [14, 23].

This work focuses on the re-execution model, for several reasons. On the one hand, replication is too costly in terms of both resource usage and energy consumption: even if the first execution turns out successful (no failure occurred), the other executions will still have to take place. Moreover, the decision of which tasks should be replicated cannot be taken when the mapping is already fixed. On the other hand, checkpointing is hard to manage with parallel processors, and too costly if there are not too many failures. Altogether, it is the "online/no-waste" characteristic of the corresponding algorithms that lead us focus on re-execution. The goal is then to ensure that each task is reliable enough, i.e., either its execution speed is above a threshold, ensuring a given reliability of the task, or the task is executed twice to enhance its reliability. There is a clear trade-off between energy consumption and reliability, since decreasing the execution speed of a task, and hence the corresponding energy consumption, is deteriorating the reliability. This calls for tackling the problem of considering the three criteria (makespan, reliability, energy)

simultaneously. This tri-criteria optimization brings dramatic complications: in addition to choosing the speed of each task, as in the deadline/energy bi-criteria problem, we also need to decide which subset of tasks should be re-executed (and then choose both execution speeds). Few authors have tackled this problem; we detail below the closest works to ours [17, 24, 1].

Izosinov et al. [17] study a tri-criteria optimization problem with a given mapping on heterogeneous architectures. However, they do not have any formal energy model, and they assume that the user specifies the maximum number of failures per processor tolerated to satisfy the reliability constraint, while we consider any number of failures but ensure a reliability threshold for each task. Zhu and Aydin [24] are also addressing a tri-criteria optimization problem similar to ours, and choose some tasks that have to be re-executed to match the reliability constraint. However, they restrict to the scheduling problem on one single processor, and they consider only the energy consumption of the first execution of a task (best-case scenario) when re-execution is done. Finally, Assayad et al. [1] have recently proposed an off-line tri-criteria scheduling heuristic (TSH), which uses active replication to minimize the makespan, with a threshold on the global failure rate and the maximum power consumption. TSH is an improved critical-path list scheduling heuristic that takes into account power and reliability before deciding which task to assign and to duplicate onto the next free processors. The complexity of this heuristic is unfortunately exponential in the number of processors. Future work will be devoted to compare our heuristics to TSH, and hence to compare re-execution with replication.

Given an application with dependence constraints and a mapping of this application on a homogeneous platform, we present in this paper theoretical results and tri-criteria heuristics that use re-execution in order to minimize the energy consumption under the constraints of both a reliability threshold per task and a deadline bound. The first contribution is a formal model for this tri-criteria scheduling problem (Section 2). The second contribution is to provide theoretical results for the different speed models, CONTINUOUS (Section 3) and VDD-HOPPING (Section 4). The third contribution is the design of novel tri-criteria scheduling heuristics that use re-execution to increase the reliability of a system under the CONTINUOUS model (Section 5), and their evaluation through extensive simulations (Section 6). To the best of our knowledge, this work is the first attempt to propose practical solutions to this tri-criteria problem. Finally, we give concluding remarks and directions for future work in Section 7.

2 The tri-criteria problem

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where $V = \{T_1, T_2, \dots, T_n\}$ is the set of tasks, $n = |V|$, and where \mathcal{E} is the set of precedence edges between tasks. For $1 \leq i \leq n$, task T_i has a weight w_i , that corresponds to the computation requirement of the task. We also consider particular class of task graphs, such as *linear chains* where $\mathcal{E} = \cup_{i=1}^{n-1} \{T_i \rightarrow T_{i+1}\}$, and *forks* with $n + 1$ tasks $\{T_0, T_1, T_2, \dots, T_n\}$ and $\mathcal{E} = \cup_{i=1}^n \{T_0 \rightarrow T_i\}$.

We assume that tasks are mapped onto a parallel platform made up of p identical processors. Each processor has a set of available speeds that is either continuous (in the interval $[f_{\min}, f_{\max}]$) or discrete (with m modes $\{f_1, \dots, f_m\}$), depending on the speed model (CONTINUOUS or VDD-HOPPING). The goal is to minimize the energy consumed during the execution of the graph while enforcing a deadline bound and matching a reliability threshold. To match the reliability threshold, some tasks are executed once at a speed high enough to satisfy the constraint, while some other tasks need to be re-executed. We detail below the conditions that are enforced on the corresponding execution speeds. The problem is therefore to decide which task to re-execute, and at which speed to run each execution of a task.

In this section, for the sake of clarity, we assume that a task is executed at the same (unique) speed throughout execution, or at two different speeds in the case of re-execution. In Section 3, we show that this strategy is indeed optimal for the CONTINUOUS model; in Section 4, we show that only two different speeds are needed for the VDD-HOPPING model (and we update the corresponding formulas accordingly). We now detail the three objective criteria (makespan, reliability, energy), and then define formally the problem.

2.1 Makespan

The makespan of a schedule is its total execution time. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. We consider a *deadline bound* D , which is a constraint on the makespan.

Let $\mathcal{E}xe(w_i, f)$ be the execution time of a task T_i of weight w_i at speed f . We assume that the cache size is adapted to the application, therefore ensuring that the execution time is linearly related to the frequency [14]: $\mathcal{E}xe(w_i, f) = \frac{w_i}{f}$. When a task is scheduled to be re-executed at two different speeds $f^{(1)}$ and $f^{(2)}$, we always account for both executions, even when the first execution is successful, and hence $\mathcal{E}xe(w_i, f^{(1)}, f^{(2)}) = \frac{w_i}{f^{(1)}} + \frac{w_i}{f^{(2)}}$. In other words, we consider a worst-case execution scenario, and the deadline D must be matched even in the case where all tasks that are re-executed fail during their first execution.

2.2 Reliability

To define the reliability, we use the fault model of Zhu et al. [25, 24]. *Transient* failures are faults caused by software errors for example. They invalidate only the execution of the current task and the processor subject to that failure will be able to recover and execute the subsequent task assigned to it (if any). In addition, we use the reliability model introduced by Shatz and Wang [21], which states that the radiation-induced transient faults follow a Poisson distribution. The parameter λ of the Poisson distribution is then:

$$\lambda(f) = \tilde{\lambda}_0 e^{\tilde{d} \frac{f_{\max} - f}{f_{\max} - f_{\min}}}, \quad (1)$$

where $f_{\min} \leq f \leq f_{\max}$ is the processing speed, the exponent $\tilde{d} \geq 0$ is a constant, indicating the sensitivity of fault rates to DVFS, and $\tilde{\lambda}_0$ is the average fault rate corresponding to f_{\max} . We see that reducing the speed for energy saving increases the fault rate exponentially. The reliability of a task T_i executed once at speed f is $R_i(f) = e^{-\lambda(f) \times \mathcal{E}xe(w_i, f)}$. Because the fault rate is usually very small, of the order of 10^{-6} per time unit in [5, 17], 10^{-5} in [1], we can use the first order approximation of $R_i(f)$ as

$$R_i(f) = 1 - \lambda(f) \times \mathcal{E}xe(w_i, f) = 1 - \tilde{\lambda}_0 e^{\tilde{d} \frac{f_{\max} - f}{f_{\max} - f_{\min}}} \times \frac{w_i}{f} = 1 - \lambda_0 e^{-df} \times \frac{w_i}{f}, \quad (2)$$

where $d = \frac{\tilde{d}}{f_{\max} - f_{\min}}$ and $\lambda_0 = \tilde{\lambda}_0 e^{df_{\max}}$. This equation holds if $\varepsilon_i = \lambda(f) \times \frac{w_i}{f} \ll 1$. With, say, $\lambda(f) = 10^{-5}$, we need $\frac{w_i}{f} \leq 10^3$ to get an accurate approximation with $\varepsilon_i \leq 0.01$: the task should execute within 16 minutes. In other words, large (computationally demanding) tasks require reasonably high processing speeds with this model (which makes full sense in practice).

We want the reliability R_i of each task T_i to be greater than a given threshold, namely $R_i(f_{\text{rel}})$, hence enforcing a local constraint dependent on the task $R_i \geq R_i(f_{\text{rel}})$. If task T_i is executed only once at speed f , then the reliability of T_i is $R_i = R_i(f)$. Since the reliability increases with speed, we must have $f \geq f_{\text{rel}}$ to match the reliability constraint. If task T_i is re-executed (speeds $f^{(1)}$ and $f^{(2)}$), then the execution of T_i is successful if and only if both attempts do not fail, so that the reliability of T_i is $R_i = 1 - (1 - R_i(f^{(1)}))(1 - R_i(f^{(2)}))$, and this quantity should be at least equal to $R_i(f_{\text{rel}})$.

2.3 Energy

The total energy consumption corresponds to the sum of the energy consumption of each task. Let E_i be the energy consumed by task T_i . For one execution of task T_i at speed f , the corresponding energy consumption is $E_i(f) = \mathcal{E}xe(w_i, f) \times f^3 = w_i \times f^2$, which corresponds to the dynamic part of the classical energy models of the literature [4, 6, 8, 3]. Note that we do not take static energy into account, because all processors are up and alive during the whole execution.

If task T_i is executed only once at speed f , then $E_i = E_i(f)$. Otherwise, if task T_i is re-executed at speeds $f^{(1)}$ and $f^{(2)}$, it is natural to add up the energy consumed during both executions, just as we add up both execution times when enforcing the makespan deadline. Again, this corresponds to the worst-case execution scenario. We obtain $E_i = E_i(f_i^{(1)}) + E_i(f_i^{(2)})$. Note that some authors [24] consider only the energy spent for the first execution, which seems unfair: re-execution comes at a price both in the deadline and in the energy consumption. Finally, the total energy consumed by the schedule, which we aim at minimizing, is $E = \sum_{i=1}^n E_i$.

2.4 Optimization problems

The two main optimization problems are derived from the two different speed models:

- TRI-CRIT-CONT. Given an application graph $\mathcal{G} = (V, \mathcal{E})$, mapped onto p homogeneous processors with continuous speeds, TRI-CRIT-CONT is the problem of deciding which tasks should be re-executed and at which speed each execution of a task should be processed, in order to minimize the total energy consumption E , subject to the deadline bound D and to the local reliability constraints $R_i \geq R_i(f_{rel})$ for each $T_i \in V$.
- TRI-CRIT-VDD. This is the same problem as TRI-CRIT-CONT, but with the VDD-HOPPING model.

We also introduce variants of the problems for particular application graphs: TRI-CRIT-CONT-CHAIN is the same problem as TRI-CRIT-CONT when the task graph is a linear chain, mapped on a single processor; and TRI-CRIT-CONT-FORK is the same problem as TRI-CRIT-CONT when the task graph is a fork, and each task is mapped on a distinct processor. We have similar definitions for the VDD-HOPPING model.

3 CONTINUOUS model

As stated in Section 2, we start by proving that with the CONTINUOUS model, it is always optimal to execute a task at a unique speed throughout its execution:

Lemma 1. *With the CONTINUOUS model, it is optimal to execute each task at a unique speed throughout its execution.*

The idea is to consider a task whose speed changes during the execution; we exhibit a speed such that the execution time of the task remains the same, but where both energy and reliability are potentially improved, by convexity of the functions.

Proof. We can assume without loss of generality that the function that gives the speed of the execution of a task is a piecewise-constant function. The proof of the general case is a direct corollary from the theorem that states that any piecewise-continuous function defined on an interval $[a, b]$ can be uniformly approximated as closely as desired by a piecewise-constant function [20].

Suppose that in the optimal solution, there is a task whose speed changes during the execution. Consider the first time-step at which the change occurs: the computation begins at speed f from time t to time t' , and then continues at speed f' until time t'' . The total energy consumption for this task in the time interval $[t, t'']$ is $E = (t' - t) \times f^3 + (t'' - t') \times (f')^3$. Moreover, the amount of work done for this task is $W = (t' - t) \times f + (t'' - t') \times f'$. The reliability of the task is exactly $1 - \lambda_0 \left((t' - t) \times e^{-df} + (t'' - t') \times e^{-df'} + r \right)$, where r is a constant due to the reliability of the rest of the process, which is independent from what happens during $[t, t'']$. The reliability is a function that increases when the function $h(t, t', t'', f, f') = (t' - t) \times e^{-df} + (t'' - t') \times e^{-df'}$ decreases.

If we run the task during the whole interval $[t, t'']$ at constant speed $f_d = W/(t'' - t)$, the same amount of work is done within the same time, and the energy consumption during this interval of time becomes $E' = (t'' - t) \times f_d^3$. Note that the new speed can be expressed as $f_d = af + (1 - a)f'$, where $0 < a = \frac{t' - t}{t'' - t} < 1$. Therefore, because of the convexity of the function $x \mapsto x^3$, we have $E' < E$. Similarly, since $x \mapsto e^{-dx}$ is a convex function, $h(t, t', t'', f_d, f_d) < h(t, t', t'', f, f')$, and the reliability constraint is also matched. This contradicts the hypothesis of optimality of the first solution, and concludes the proof. \square

Next we show that not only a task is executed at a single speed, but that its re-execution (whenever it occurs) is executed at the same speed as its first execution:

Lemma 2. *With the CONTINUOUS model, it is optimal to re-execute each task (whenever needed) at the same speed as its first execution, and this speed f is such that $f_i^{(inf)} \leq f < \frac{1}{\sqrt{2}} f_{rel}$, where*

$$\lambda_0 w_i \frac{e^{-2df_i^{(inf)}}}{(f_i^{(inf)})^2} = \frac{e^{-df_{rel}}}{f_{rel}}. \quad (3)$$

Similarly to the proof of Lemma 1, we exhibit a unique speed for both executions, in case they differ, so that the execution time remains identical but both energy and reliability are improved. If this unique speed is greater than $\frac{1}{\sqrt{2}}f_{\text{rel}}$, then it is better to execute the task only once at speed f_{rel} , and if f is lower than $f_i^{(\text{inf})}$, then the reliability constraint is not matched.

Proof. Consider a task T_i executed a first time at speed f_i , and a second time at speed $f'_i > f_i$. Assume first that $d = 0$, i.e., the reliability of task T_i executed at speed f_i is $R_i(f_i) = 1 - \lambda_0 \frac{w_i}{f_i}$. We show that executing task T_i twice at speed $f = \sqrt{f_i f'_i}$ improves the energy consumption while matching the deadline and reliability constraints. Clearly the reliability constraint is matched, since $1 - \lambda_0 w_i^2 \frac{1}{f^2} = 1 - \lambda_0 w_i^2 \frac{1}{f_i f'_i}$. The fact that the deadline constraint is matched is due to the fact that $\sqrt{f_i f'_i} \geq \frac{2f_i f'_i}{f_i + f'_i}$ (by squaring both sides of the equation we obtain $(f_i - f'_i)^2 \geq 0$). Then we use the fact that $f_d = \frac{2f_i f'_i}{f_i + f'_i}$ is the minimal speed such that $\forall f \geq f_d, \frac{2w_i}{f} < \frac{w_i}{f_i} + \frac{w_i}{f'_i}$. Finally, it is easy to see that the energy consumption is improved since $2f_i f'_i \leq f_i^2 + f'^2_i$, hence $2w_i f_i f'_i \leq w_i f_i^2 + w_i f'^2_i$.

In the general case when $d \neq 0$, instead of having a closed form formula for the new speed f common to both executions, we have $f = \max(f_1, f_2)$, where f_1 is dictated by the reliability constraint, while f_2 is dictated by the deadline constraint. f_1 is the solution to the equation $2(dX + \ln X) = (df_i + \ln f_i) + (df'_i + \ln f'_i)$; this equation comes from the reliability constraint: the minimum speed X to match the reliability is obtained with $1 - \lambda_0^2 w_i^2 \frac{e^{-df_i}}{f_i} \frac{e^{-df'_i}}{f'_i} = 1 - \lambda_0^2 w_i^2 \frac{e^{-2dX}}{X^2}$. The deadline constraint must also be enforced, and hence $f_2 = \frac{2f_i f'_i}{f_i + f'_i}$ (minimum speed to match the deadline). Then the fact that the energy does not increase comes from the convexity of this function.

Let f be the unique speed at which the task is executed (twice). If $f \geq \frac{1}{\sqrt{2}}f_{\text{rel}}$, then executing the task only once at speed f_{rel} has a lower energy consumption and execution time, while still matching the reliability constraint. Hence it is not optimal to re-execute the task unless $f < \frac{1}{\sqrt{2}}f_{\text{rel}}$. Finally, note that f must be greater than $f_i^{(\text{inf})}$, solution of Equation (3), since $f_i^{(\text{inf})}$ is the minimum speed such that the reliability constraint is met if task T_i is executed twice at the same speed. \square

Note that both lemmas can be applied to any solution of the TRI-CRIT-CONT problem, not just optimal solutions, hence all heuristics of Section 5 will assign a unique speed to each task, be it re-executed or not.

We are now ready to assess the problem complexity:

Theorem 1. *The TRI-CRIT-CONT-CHAIN problem is NP-hard, but not known to be in NP.*

Note that the problem is not known to be in NP because speeds could take any real values (CONTINUOUS model). The completeness comes from SUBSET-SUM [11]. The problem is NP-hard even for a linear chain application mapped on a single processor (and any general DAG mapped on a single processor becomes a linear chain).

Proof. Consider the associated decision problem: given a deadline, and energy and reliability bounds, can we schedule the graph to match all these bounds? Since the speeds could take any real values, the problem is not known to be in NP. For the completeness, we use a reduction from SUBSET-SUM [11]. Let \mathcal{I}_1 be an instance of SUBSET-SUM: given n strictly positive integers a_1, \dots, a_n , and a positive integer X , does there exist a subset I of $\{1, \dots, n\}$ such that $\sum_{i \in I} a_i = X$? Let $S = \sum_{i=1}^n a_i$.

We build the following instance \mathcal{I}_2 of our problem. The execution graph is a linear chain with n tasks, where:

- task T_i has weight $w_i = a_i$;
- $\lambda_0 = \frac{f_{\text{max}}}{100 \max_i a_i}$;
- $f_{\text{min}} = \sqrt{\lambda_0 \max_i a_i f_{\text{max}}} = \frac{1}{10} f_{\text{max}}$;
- $f_{\text{rel}} = f_{\text{max}}$; $d = 0$.

The bounds on reliability, deadline and energy are:

- $R_i^0 = R_i(f_{\text{rel}}) = 1 - \lambda_0 \frac{w_i}{f_{\text{rel}}}$ for $1 \leq i \leq n$;
- $D_0 = \frac{S}{f_{\text{rel}}} + \frac{X}{cf_{\text{rel}}}$, where c is the unique positive real root of the polynomial $7y^3 + 21y^2 - 3y - 1$. Analytically, we derive that $c = 4\sqrt{\frac{2}{7}} \cos \frac{1}{3}(\pi - \tan^{-1} \frac{1}{\sqrt{7}}) - 1$ (≈ 0.2838); but this value is irrational, so have to we encode it symbolically rather than numerically;

- $E_0 = 2X\left(\frac{2c}{1+c}f_{\text{rel}}\right)^2 + (S-X)f_{\text{rel}}^2$.

Clearly, the size of \mathcal{I}_2 is polynomial in the size of \mathcal{I}_1 .

Suppose first that instance \mathcal{I}_1 has a solution, I . For all $i \in I$, T_i is executed twice at speed $\frac{2c}{1+c}f_{\text{rel}}$. Otherwise, for all $i \notin I$, it is executed only once at speed f_{rel} . The execution time is $\sum_{i \notin I} \frac{a_i}{f_{\text{rel}}} + \sum_{i \in I} 2\frac{a_i}{\frac{2c}{1+c}f_{\text{rel}}} = \frac{S-X}{f_{\text{rel}}} + 2X\frac{1+c}{2cf_{\text{rel}}} = D_0$. The reliability constraint is obviously met for tasks not in I . It is also met for all tasks in I , since $\frac{2c}{1+c}f_{\text{rel}} > f_{\text{min}}$, and two executions at f_{min} are sufficient to match the reliability constraint. Indeed, $1 - \lambda_0^2 \frac{a_i^2}{f_{\text{min}}^2} = 1 - \lambda_0 \frac{a_i}{f_{\text{rel}}} \frac{a_i}{\max_i a_i} \geq 1 - \lambda_0 \frac{a_i}{f_{\text{rel}}} = R_i^0$. The energy consumption is exactly E_0 . All bounds are respected, and therefore we have a solution to \mathcal{I}_2 .

Suppose now that \mathcal{I}_2 has a solution. Let $I = \{i \mid T_i \text{ is executed twice in the solution}\}$, and $Y = \sum_{i \in I} a_i$. We prove in the following that necessarily $Y = X$, since the energy constraint E_0 is respected in \mathcal{I}_2 .

We first point out that tasks executed only once are necessarily executed at maximum speed to match the reliability constraint. Then consider the problem of minimizing the energy of a set of tasks, some executed twice, some executed once at maximum speed, and assume that we have a deadline D_0 to match, but no constraint on reliability or on f_{min} . We will verify later that these additional two constraints are indeed satisfied by the optimal solution when the only constraint is the deadline. Thanks to Lemma 2, for all $i \in I$, task T_i is executed twice at the same speed. It is easy to see that in fact all tasks in I are executed at the same speed, otherwise we could decrease the energy consumption without modifying the execution time, by convexity of the function. Let f be the speed of execution (and re-execution) of task T_i , with $i \in I$. Because the deadline is the only constraint, either $Y = 0$ (no tasks are re-executed), or it is optimal to exactly match the deadline D_0 (otherwise we could just slow down all the re-executed tasks and this would decrease the total energy). Hence the problem amounts to find the values of Y and f that minimize the function $E = 2Yf^2 + (S-Y)f_{\text{rel}}^2$, with the constraint $(S-Y)/f_{\text{rel}} + 2Y/f \leq D_0$. First, note that if $Y = 0$ then $E > E_0$, and hence $Y > 0$ (since it corresponds to a solution of \mathcal{I}_2). Therefore, since the deadline is tight, we have $f = \frac{2Y}{D_0f_{\text{rel}} - (S-Y)}f_{\text{rel}}$, and finally the energy consumption can be expressed as

$$E(Y) = \left(\frac{(2Y)^3}{(D_0f_{\text{rel}} - (S-Y))^2} + (S-Y) \right) f_{\text{rel}}^2.$$

We aim at finding the minimum of this function. Let $\tilde{Y} = \frac{Y}{D_0f_{\text{rel}} - S}$. Then we have $E(\tilde{Y}) = \left(\frac{(2\tilde{Y})^3}{(1+\tilde{Y})^2} + \left(\frac{S}{D_0f_{\text{rel}} - S} - \tilde{Y} \right) \right) \times (D_0f_{\text{rel}} - S)f_{\text{rel}}^2$. Differentiating, we obtain

$$E'(\tilde{Y}) = \left(\frac{3 \times 2^3 \tilde{Y}^2}{(1+\tilde{Y})^2} - \frac{2^4 \tilde{Y}^3}{(1+\tilde{Y})^3} - 1 \right) (D_0f_{\text{rel}} - S)f_{\text{rel}}^2.$$

Finally, $E'(\tilde{Y}) = 0$ if and only if

$$24\tilde{Y}^2(1+\tilde{Y}) - 16\tilde{Y}^3 - (1+\tilde{Y})^3 = 0. \quad (4)$$

The only positive solution of Equation (4) is $\tilde{Y} = c$, and therefore the unique minimum of $E(Y)$ is obtained for $Y = c(D_0f_{\text{rel}} - S) = X$.

Note that for $Y = X$, we have $E = E_0$, and therefore any other value of Y would not correspond to a solution. There remains to check that the solution matches both constraints on f_{min} and on reliability, to confirm the hypothesis on the speed of tasks that are re-executed. Using the same argument as in the first part of the proof, we see that the reliability constraint is respected when a task is executed twice at f_{min} , and therefore we just need to check that $f \geq f_{\text{min}}$. For $Y = X$, we have $f = \frac{2c}{1+c}f_{\text{rel}} > f_{\text{min}}$.

Altogether, we have $\sum_{i \in I} a_i = Y = X$, and therefore \mathcal{I}_1 has a solution. This concludes the proof. \square

Even if TRI-CRIT-CONT-CHAIN is NP-hard, we can characterize an optimal solution of the problem:

Proposition 1. *If $f_{\text{rel}} < f_{\text{max}}$, then in any optimal solution of TRI-CRIT-CONT-CHAIN, either all tasks are executed only once, at constant speed $\max(\frac{\sum_{i=1}^n w_i}{D}, f_{\text{rel}})$; or at least one task is re-executed, and then all tasks that are not re-executed are executed at speed f_{rel} .*

Proof. Consider an optimal schedule. If all tasks are executed only once, the smallest energy consumption is obtained when using the constant speed $\frac{\sum_{i=1}^n w_i}{D}$. However if $\frac{\sum_{i=1}^n w_i}{D} < f_{\text{rel}}$, then we have to execute all tasks at speed f_{rel} to match both reliability and deadline constraints.

Now, assume that some task T_i is re-executed, and assume by contradiction, that some other task T_j is executed only once at speed $f_j > f_{\text{rel}}$. Note that the common speed f_i used in both executions of T_i is smaller than f_{rel} , otherwise we would not need to re-execute T_i . We have $f_i < f_{\text{rel}} < f_j$, and we prove that there exist values f'_i (new speed of one execution of T_i) and f'_j (new speed of T_j) such that $f_i < f'_i$, $f_{\text{rel}} \leq f'_j < f_j$, and the energy consumed with the new speeds is strictly smaller, while the execution time is unchanged. The constraint on reliability will also be met, since the speed of one execution of T_i is increased, while the speed of T_j remains above the reliability threshold. Note that we do not modify the speed of the re-execution of T_i (that remains f_i), and the time and energy consumption of this execution are not accounted for in the equations. Also, we restrict to values such that $f'_i \leq f'_j$.

Our problem writes: do there exist $\epsilon, \epsilon' > 0$ such that

$$\begin{aligned} w_i f_i^2 + w_j f_j^2 &> w_i (f_i + \epsilon')^2 + w_j (f_j - \epsilon)^2; \\ D &= \frac{w_i}{f_i} + \frac{w_j}{f_j} = \frac{w_i}{f_i + \epsilon'} + \frac{w_j}{f_j - \epsilon}; \\ f_i &< f_i + \epsilon' \leq f_j - \epsilon; \\ f_{\text{rel}} &\leq f_j - \epsilon < f_j. \end{aligned}$$

We study the function $\phi : \epsilon \mapsto w_i f_i^2 + w_j f_j^2 - (w_i (f_i + \epsilon')^2 + w_j (f_j - \epsilon)^2)$, and we want to prove that it is positive. Thanks to the deadline constraint (D is the bound on the execution time of T_j plus one execution of T_i), we have $f_i = \frac{w_i f_j}{D f_j - w_j}$, and $f_i + \epsilon' = \frac{w_i (f_j - \epsilon)}{D - \frac{w_j}{f_j - \epsilon}} = \frac{w_i (f_j - \epsilon)}{D(f_j - \epsilon) - w_j}$.

We can therefore express $\phi(\epsilon)$ as:

$$\phi(\epsilon) = \frac{w_i^3 f_j^2}{(D f_j - w_j)^2} - \frac{w_i^3 (f_j - \epsilon)^2}{(D(f_j - \epsilon) - w_j)^2} + w_j f_j^2 - w_j (f_j - \epsilon)^2.$$

Moreover, we study the function for $\epsilon > 0$, and because of the constraint on new speeds, $\epsilon \leq f_j - f_{\text{rel}}$. Another bound on ϵ is obtained from the fact that $f_i + \epsilon' \leq f_j - \epsilon$, and the equality is obtained when both tasks are running at speed $\frac{w_i + w_j}{D}$, thus meeting the deadline. Hence, $f_j - \epsilon \geq \frac{w_i + w_j}{D}$, and finally

$$0 < \epsilon \leq f_j - \max\left(f_{\text{rel}}, \frac{w_i + w_j}{D}\right).$$

Differentiating, we obtain

$$\phi'(\epsilon) = \frac{2w_i^3 (f_j - \epsilon)}{(D(f_j - \epsilon) - w_j)^2} - \frac{2Dw_i^3 (f_j - \epsilon)^2}{(D(f_j - \epsilon) - w_j)^3} + 2w_j (f_j - \epsilon).$$

We are looking for ϵ such that $\phi'(\epsilon) = 0$, hence obtaining the polynomial

$$X^3 - \frac{w_i^3}{w_j^3} = 0,$$

by multiplying each side of the equation by $\frac{(D(f_j - \epsilon) - w_j)^3}{w_j^3 (f_j - \epsilon)^3}$, and defining $X = \frac{D(f_j - \epsilon) - w_j}{w_j}$. The only real solution to this polynomial is $X = \frac{w_i}{w_j}$, that corresponds to $\epsilon = f_j - \frac{w_i + w_j}{D}$. Therefore, the only extremum of the function ϕ is

obtained for this value of ϵ , which corresponds to executing both tasks at the same speed. Because of the convexity of the energy consumption, this value corresponds to a maximum of function ϕ (see for instance Proposition 2 in [3]), since the energy is minimized when both tasks run at the same speed. Therefore, ϕ is strictly increasing for $0 \leq \epsilon \leq f_j - \frac{w_i + w_j}{D}$, and for $\epsilon = f_j - \max\left(f_{\text{rel}}, \frac{w_i + w_j}{D}\right)$, ϕ is maximal (with regards to our constraints), and $\phi(\epsilon) > 0$.

Altogether, this value of ϵ gives us two new speeds $f'_i = \frac{w_i(f_j - \epsilon)}{D(f_j - \epsilon) - w_j}$ and $f'_j = f_j - \epsilon$ that strictly improve the energy consumption of the schedule, while the constraints on deadline and reliability are still enforced. However, the original schedule was supposed to be optimal, we have a contradiction, which concludes the proof. \square

In essence, Proposition 1 states that when dealing with a linear chain, we should first slow down the execution of each task as much as possible. Then, if the deadline is not too tight, i.e., if $f_{\text{rel}} > \frac{\sum_{i=1}^n w_i}{D}$, there remains the possibility to re-execute some of the tasks (and of course it is NP-hard to decide which ones). Still, this general principle “*first slow-down and then re-execute*” will guide the design of type A heuristics in Section 5.

While the general TRI-CRIT-CONT problem is NP-hard even with a single processor, the particular variant TRI-CRIT-CONT-FORK can be solved in polynomial time:

Theorem 2. *The TRI-CRIT-CONT-FORK problem can be solved in polynomial time.*

The difficulty to provide an optimal algorithm for the TRI-CRIT-CONT-FORK problem comes from the fact that the total execution time must be shared between the source of the fork, T_0 , and the other tasks that all run in parallel. If we know D' , the fraction of the deadline allotted for tasks T_1, \dots, T_n once the source has finished its execution, then we can decide which tasks are re-executed and all execution speeds.

Proof. We start by showing that TRI-CRIT-CONT can be solved in polynomial time for one single task, and then for n independent tasks, before tackling the problem TRI-CRIT-CONT-FORK.

TRI-CRIT-CONT for a single task on one processor can be solved in polynomial time. When there is a single task T of weight w , the solution depends on the deadline D :

1. if $D < \frac{w}{f_{\text{max}}} = D^{(0)}$, then there is no solution;
2. if $\frac{w}{f_{\text{max}}} \leq D \leq \frac{w}{f_{\text{rel}}} = D^{(1)}$, then T is executed once at speed $\frac{w}{D}$, the minimum energy is $w^3 \times \frac{1}{D^2}$;
3. if $\frac{w}{f_{\text{rel}}} < D \leq \frac{2\sqrt{2}w}{f_{\text{rel}}} = D^{(2)}$, then T is executed once at speed f_{rel} , the minimum energy is wf_{rel}^2 ;
4. if $\frac{2\sqrt{2}w}{f_{\text{rel}}} < D \leq \frac{2w}{f^{(\text{inf})}} = D^{(3)}$, then T is executed twice at speed $\frac{2w}{D}$, the minimum energy is $(2w)^3 \times \frac{1}{D^2}$;
5. if $\frac{2w}{f^{(\text{inf})}} < D$, then T is executed twice at speed $f^{(\text{inf})}$, the minimum energy is $2wf^{(\text{inf})2}$.

These results are a direct consequence from the deadline and reliability constraints. With a deadline smaller than $D^{(0)}$, the task cannot be executed within the deadline, even at speed f_{max} . The bound $D^{(2)}$ comes from Lemma 2, which states that we need to have enough time to execute the task twice at a speed lower than $\frac{1}{\sqrt{2}}f_{\text{rel}}$ before re-executing it. Therefore, the task is executed only once for smaller deadlines, either at speed w/D , or at speed f_{rel} if $w/D < f_{\text{rel}}$. For larger deadlines, the task is re-executed, either at speed $2w/D$, or at speed $f^{(\text{inf})}$ if $2w/D < f^{(\text{inf})}$, since the re-execution speed cannot be lower than $f^{(\text{inf})}$ (see Lemma 2).

TRI-CRIT-CONT for n independent tasks on n processors can be solved in polynomial time. For n independent tasks mapped on n distinct processors, decisions for each task can be made independently, and we simply solve n times the previous single task problem. The minimum energy is the sum of the minimum energies obtained for each task.

TRI-CRIT-CONT-FORK. For a fork, we need to decide how to share the deadline between the source T_0 of the fork and the other tasks (i.e., n independent tasks on n processors). We search the optimal values D_1 and D_2 such that $D_1 + D_2 = D$, and the energy of executing T_0 within deadline D_1 plus the energy of executing all other tasks within D_2 is minimum. Therefore, we just need to find the optimal value for D_2 (since $D_1 = D - D_2$), and reuse previous results for independent tasks.

Independently of D , we can define for each task T_i four values $D_i^{(0)}, D_i^{(1)}, D_i^{(2)}$ and $D_i^{(3)}$, as in the case of a single task. There is a solution if and only if $\max_{1 \leq i \leq n} D_i^{(0)} \leq D_2 \leq D - D_0^{(0)}$. Then, the energy consumption depends upon the intervals delimited by values $D - D_0^{(j)}$ and $D_i^{(j)}$, for $1 \leq i \leq n$ and $j = 1, 2, 3$. Within an interval, the energy consumed by the source is either a constant, or a constant times $\frac{1}{(D-D_2)^2}$, and the energy consumed by task T_i ($1 \leq i \leq n$) is either a constant, or a constant times $\frac{1}{D_2^2}$. All the constants are known, only dependent of T_i , and they are obtained by the algorithm that gives the optimal solution to TRI-CRIT-CONT for a single task. To obtain the intervals, we sort the $4n$ values of $D_i^{(j)}$ ($i > 0$) and the four values of $D - D_0^{(j)}$, with $j = 0, 1, 2, 3$, and rename these $4(n+1)$ values as d_k , with $1 \leq k \leq 4(n+1)$ and $d_k \leq d_{k+1}$. Given the bounds on D_2 , we consider the intervals of the form $[d_k, d_{k+1}]$, with $d_k \geq \max_{1 \leq i \leq n} D_i^{(0)}$, and $d_{k+1} \leq D - D_0^{(0)}$. On each of these intervals, the energy function is $\frac{K}{(D-D_2)^2} + \frac{K'}{D_2^2} + K''$, where K, K' and K'' are positive constants that can be obtained in polynomial time by the solution to TRI-CRIT-CONT for a single task. Finding a minimum to this function on the interval $[d_k, d_{k+1}]$ can be done in polynomial time:

- the first derivative of this function is $\frac{2K}{(D-D_2)^3} - \frac{2K'}{D_2^3}$;
- the function is convex on $]0, D[$, indeed the second derivative of this function is $\frac{6K}{(D-D_2)^4} + \frac{6K'}{D_2^4}$, which is positive on $]0, D[$, and therefore on the interval $[d_k, d_{k+1}]$, there is exactly one minimum to the energy function ($d_k > 0$ and $d_{k+1} < D$);
- the minimum is obtained either when the first derivative is equal to zero in the interval (i.e., if there is a solution to the equation $2KD_2^3 - 2K'(D - D_2)^3 = 0$ in $[d_k, d_{k+1}]$), or the minimum is reached at d_k (resp. d_{k+1}) if the first derivative is positive (resp. negative) on the interval.

There are $O(n)$ intervals, and it takes constant time to find the minimum energy E_k within interval $[d_k, d_{k+1}]$, as explained above, by solving one equation. Since we have partitioned the interval of possible deadlines $D_2 \in [\max_{1 \leq i \leq n} D_i^{(0)}, D - D_0^{(0)}]$, and obtained the minimum energy consumption in each sub-interval, the minimum energy consumption for the fork graph is $\min_k E_k$, and the value of D_2 is obtained where the minimum is reached. Once we know the optimal value of D_2 , it is easy to reconstruct the solution, following the algorithm for a single task, in polynomial time. \square

Note that this algorithm does not provide any closed-form formula for the speeds of the tasks, and that there is an intricate case analysis due to the reliability constraints.

If we further assume that the fork is made of identical tasks (i.e., $w_i = w$ for $0 \leq i \leq n$), then we can provide a closed-form formula. However, Proposition 2 illustrates the inherent difficulty of this *simple* problem, with several cases to consider depending on the values of the deadline, and also the bounds on speeds ($f_{\min}, f_{\max}, f_{\text{rel}}$, etc.). First, since the tasks all have the same weight $w_i = w$, we get rid of the $f_i^{(\text{inf})}$ introduced above, since they are all identical (see Equation (3)): $f_i^{(\text{inf})} = f^{(\text{inf})}$ for $0 \leq i \leq n$. Therefore we let $f_{\min} = \max(f_{\min}, f^{(\text{inf})})$ in the proposition below:

Proposition 2. *In the optimal solution of TRI-CRIT-CONT-FORK with at least three identical tasks (and hence $n \geq 2$), there are only three possible scenarios: (i) no task is re-executed; (ii) the n successors are all re-executed but not the source; (iii) all tasks are re-executed. In each scenario, the source is executed at speed f_{src} (once or twice), and the n successors are executed at the same speed f_{leaf} (once or twice).*

For a deadline $D < \frac{2w}{f_{\max}}$, there is no solution. For a deadline $D \in \left[\frac{2w}{f_{\max}}, \frac{w}{f_{\text{rel}}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} \right]$, no task is re-executed (scenario (i)) and the values of f_{src} and f_{leaf} are the following:

- if $\frac{2w}{f_{\max}} \leq D \leq \min\left(\frac{w}{f_{\max}}(1+n^{\frac{1}{3}}), w(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\max}})\right)$, then $f_{\text{src}} = f_{\max}$ and $f_{\text{leaf}} = \frac{w}{Df_{\max}-w}f_{\max}$;
- if $\frac{w}{f_{\max}}(1+n^{\frac{1}{3}}) \leq w(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\max}})$, then
 - if $\frac{w}{f_{\max}}(1+n^{\frac{1}{3}}) < D \leq \frac{w}{f_{\text{rel}}}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, then $f_{\text{src}} = \frac{w}{D}(1+n^{\frac{1}{3}})$ and $f_{\text{leaf}} = \frac{w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
 - if $\frac{w}{f_{\text{rel}}}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{\text{rel}}}$, then $f_{\text{src}} = \frac{w}{Df_{\text{rel}}-w}f_{\text{rel}}$ and $f_{\text{leaf}} = f_{\text{rel}}$;
- if $\frac{w}{f_{\max}}(1+n^{\frac{1}{3}}) > w(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\max}})$, then
 - if $w(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\max}}) < D \leq \frac{2w}{f_{\text{rel}}}$, then $f_{\text{src}} = \frac{w}{Df_{\text{rel}}-w}f_{\text{rel}}$ and $f_{\text{leaf}} = f_{\text{rel}}$;
- if $\frac{2w}{f_{\text{rel}}} < D \leq \frac{w}{f_{\text{rel}}}\frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$, then $f_{\text{src}} = f_{\text{leaf}} = f_{\text{rel}}$.

Note that for larger values of D , depending on f_{\min} , we can move to scenarios (ii) and (iii) with partial or total re-execution. The case analysis becomes even more painful, but remains feasible. Intuitively, the property that all tasks have the same weight is the key to obtaining analytical formulas, because all tasks have the same minimum speed $f^{(\text{inf})}$ dictated by Equation (3).

Proof. First, we recall preliminary results:

- if a task is executed only once at speed f , then $f_{\text{rel}} \leq f \leq f_{\max}$;
- if a task is re-executed, then both executions are done at the same speed f , and $f_{\min} \leq f < \frac{1}{\sqrt{2}}f_{\text{rel}}$.

By hypothesis, all tasks are identical: the bound on re-execution speed accounts for $f^{(\text{inf})}$ as in Lemma 2, since we now have $f_{\min} = \max(f_{\min}, f^{(\text{inf})})$. Therefore, if two tasks of same weight w have the same energy consumption in the optimal solution, then they are executed the same number of times (once or twice) and at the same speed(s). If the energy is greater than or equal to wf_{rel}^2 , then necessarily there is one execution; and if it is lower than wf_{rel}^2 , then necessarily there are two executions.

First, we prove that in any solution, the energy consumed for the execution of each successor task, also called *leaf*, is the same. If it was not the case, since each task has the same weight, and since each leaf is independent from the other and only dependent on the source of the fork, if a leaf T_i is consuming more than another leaf T_j , then we could execute T_i the same number of times and at the same speed than T_j , hence matching the deadline bound and the reliability constraint, and obtaining a better solution. Thanks to this result, we now assume that all leaves are executed at the same speed(s), denoted f_{leaf} . The source task may be executed at a different speed, f_{src} .

Next, let us show that the energy consumption of the source is always greater than or equal to that of any leaf in any optimal solution. First, since the source and leaves have the same weight, if we invert the execution speeds of the source and of the leaves, then the reliability of each task is still matched, and so is the execution time. Moreover, the energy consumption is equal to the energy consumption of the source plus n times the energy consumption of any leaf (recall that they all consume the same amount of energy). Hence, if the energy consumption of the source is smaller than the one of the leaves, permuting those execution speeds would reduce by $(n-1) \times \Delta$ the energy, where Δ is the positive difference between the two energy consumptions. Thanks to this result, we can say that the source should never be executed twice if the leaves are executed only once since it would mean a lower energy consumption for the source (recall that $n \geq 2$).

This result fully characterizes the shape of any optimal solution. There are only three possible scenarios: (i) no task is re-executed; (ii) the n successors (leaves) are all re-executed but not the source; (iii) all tasks are re-executed. We study independently the three scenarios, i.e., we aim at determining the values of f_{src} and f_{leaf} in each case. Conditions on the deadline indicate the shape of the solution, and we perform the case analysis for deadlines

$$D \leq \frac{w}{f_{\text{rel}}}\frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}.$$

Let us assume first that the optimal solution is such that each task is executed only once (scenario (i)). From the proof of Theorem 1 in [3], we obtain the optimal speeds with no re-execution and without accounting for reliability; they are given by the following formulas:

- if $D < \frac{2w}{f_{\max}}$, then there is no solution, since the tasks executed at f_{\max} exceed the deadline;
- if $\frac{2w}{f_{\max}} \leq D \leq \frac{w}{f_{\max}}(1+n^{\frac{1}{3}})$, then $f_{\text{src}} = f_{\max}$ and $f_{\text{leaf}} = \frac{w}{Df_{\max}-w}f_{\max}$;
- if $\frac{w}{f_{\max}}(1+n^{\frac{1}{3}}) < D$, then $f_{\text{src}} = \frac{w}{D}(1+n^{\frac{1}{3}})$ and $f_{\text{leaf}} = \frac{w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$.

Since there is a minimum speed f_{rel} to match the reliability constraint, there is a condition when $f_{\text{leaf}} < f_{\text{rel}}$ that makes an amendment on some of the items. Note that in all cases, if $D > \frac{2w}{f_{\text{rel}}}$, then both the source and the leaves are executed at speed f_{rel} , i.e., $f_{\text{src}} = f_{\text{leaf}} = f_{\text{rel}}$ (recall that we consider the case with no re-execution).

- If $\frac{2w}{f_{\text{max}}} \leq D \leq \frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}})$, then we need $f_{\text{leaf}} = \frac{w}{Df_{\text{max}}-w}f_{\text{max}} \geq f_{\text{rel}}$, hence the condition: $D \leq \min\left(\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}), w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right)\right)$. In this case, $f_{\text{src}} = f_{\text{max}}$ and $f_{\text{leaf}} = \frac{w}{Df_{\text{max}}-w}f_{\text{max}}$.
- If $D > \min\left(\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}), w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right)\right)$, then the previous results do not hold anymore because of the constraint on the speed of the leaves. We must further differentiate cases, depending on where the minimum is reached.
- If $\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}) \leq w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right)$, then
 - if $\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}) < D \leq \frac{w}{f_{\text{rel}}}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, we are in the third case with no reliability, and therefore $f_{\text{src}} = \frac{w}{D}(1 + n^{\frac{1}{3}})$ and $f_{\text{leaf}} = \frac{w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$; the upper bound on D guarantees that $f_{\text{leaf}} \geq f_{\text{rel}}$, while the lower bound on D guarantees that $f_{\text{src}} \leq f_{\text{max}}$;
 - if $\frac{w}{f_{\text{rel}}}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{\text{rel}}}$, then the speed of the leaves is constrained by f_{rel} , and we obtain $f_{\text{leaf}} = f_{\text{rel}}$ and $f_{\text{src}} = \frac{w}{Df_{\text{rel}}-w}f_{\text{rel}}$. From the lower bound on D , we obtain $f_{\text{src}} < n^{\frac{1}{3}}f_{\text{rel}}$, and since $\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}) \leq w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right)$, we have $f_{\text{src}} < n^{\frac{1}{3}}f_{\text{rel}} \leq f_{\text{max}}$.
- If $\frac{w}{f_{\text{max}}}(1 + n^{\frac{1}{3}}) > w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right)$, then for $w\left(\frac{1}{f_{\text{rel}}} + \frac{1}{f_{\text{max}}}\right) < D \leq \frac{2w}{f_{\text{rel}}}$, the leaves should be executed at speed $f_{\text{leaf}} = f_{\text{rel}}$, and for the source, $f_{\text{src}} = \frac{w}{Df_{\text{rel}}-w}f_{\text{rel}}$. Note that the lower bound on D is equivalent to $\frac{w}{Df_{\text{rel}}-w}f_{\text{rel}} < f_{\text{max}}$, and hence the speed of the source is not exceeding f_{max} .

As stated above, if $D > \frac{2w}{f_{\text{rel}}}$, both the source and the leaves are executed at speed f_{rel} (with no re-execution). However, if the deadline is larger, re-execution will be used by the optimal solution (i.e., it will become scenario (ii)). Let us consider therefore the scenario in which leaves are re-executed, to compare the energy consumption with the first scenario. In this case, we consider an equivalent fork in which leaves are of weight $2w$, and a schedule with no re-execution. Then the optimal solution when there is no maximum speed is:

$$f_{\text{src}} = \frac{w}{D}(1 + 2n^{\frac{1}{3}}) \quad \text{and} \quad f_{\text{leaf}} = \frac{w}{D}\frac{1 + 2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}.$$

If $f_{\text{leaf}} \geq \frac{1}{\sqrt{2}}f_{\text{rel}}$, then there is a better solution to the original problem without re-execution. Indeed, the solution in which the leaves (of weight w) are executed once at speed $f'_{\text{leaf}} = \max(f_{\text{leaf}}, f_{\text{rel}})$ is such that:

- the reliability constraint is matched ($f'_{\text{leaf}} \geq f_{\text{rel}}$);
- the deadline constraint is matched ($f'_{\text{leaf}} \geq f_{\text{leaf}}$, and f_{leaf} corresponds to the solution with re-execution, i.e., $w/f_{\text{src}} + 2w/f_{\text{leaf}} \leq D$);
- the energy consumption is better, as stated by Lemma 2 if $f'_{\text{leaf}} = f_{\text{rel}}$.

Therefore, we are in scenario (ii) when $f_{\text{leaf}} < \frac{1}{\sqrt{2}}f_{\text{rel}}$, i.e., $D > \frac{w}{f_{\text{rel}}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$.

Moreover, depending whether $f_{\text{src}} \geq f_{\text{rel}}$ or $f_{\text{src}} < f_{\text{rel}}$:

- if $f_{\text{src}} \geq f_{\text{rel}}$, i.e., $D \leq \frac{w}{f_{\text{rel}}}(1 + 2n^{\frac{1}{3}})$, then the solution is valid;
- if $f_{\text{src}} < f_{\text{rel}}$, then we must in fact have $f_{\text{src}} = f_{\text{rel}}$, and then $f_{\text{leaf}} = \max(\frac{2w}{Df_{\text{rel}}-w}f_{\text{rel}}, f_{\text{min}})$.

Note that these values do not take into account the constraints f_{max} and f_{min} . Therefore, they are lower bounds on the energy consumption when the leaves are re-executed.

Finally, we establish a bound D_0 on the deadline: for larger values than D_0 , we cannot guarantee that re-execution will not be used by the optimal solution, and hence we will have fully characterized the cases for deadlines smaller than D_0 . Since we have only computed lower bounds on energy consumption for the scenario (ii), this bound will not be tight. We know that the minimum energy consumption is a function decreasing with the deadline: if $D > D'$, then any solution for D' is a solution for D . Let us find the minimum deadline D such that the energy when the leaves are re-executed is smaller than the energy when no task is re-executed.

As we have seen before, necessarily if $D \leq \frac{w}{f_{\text{rel}}} \sqrt{2} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, then it is better to have no re-execution, i.e., $D_0 \geq \frac{w}{f_{\text{rel}}} \sqrt{2} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$. Let $D = \frac{w}{f_{\text{rel}}} \sqrt{2} \frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} + \epsilon$. We suppose also that $D \leq \frac{w}{f_{\text{rel}}} (1 + 2n^{\frac{1}{3}})$, i.e., the solution with re-execution is valid ($f_{\text{src}} \geq f_{\text{rel}}$).

- The energy consumption when the leaves are re-executed is greater than

$$E_2 = wf_{\text{src}}^2 + 2nw f_{\text{leaf}}^2 = \frac{w^3}{D^2} (1 + 2n^{\frac{1}{3}})^3.$$

- With no re-execution, the deadline is large enough so that each task can be executed at speed f_{rel} , and therefore the energy consumption is

$$E_1 = (1+n)wf_{\text{rel}}^2 = 2 \frac{w^3}{(D-\epsilon)^2} (1+n) \left(\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \right)^2.$$

We now check the condition $E_1 \leq E_2$:

$$\begin{aligned} 2 \frac{w^3}{(D-\epsilon)^2} (1+n) \left(\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \right)^2 &\leq \frac{w^3}{D^2} (1+2n^{\frac{1}{3}})^3 \\ \frac{2}{(D-\epsilon)^2} \frac{1+n}{n^{\frac{1}{3}}} &\leq \frac{1+2n^{\frac{1}{3}}}{D^2} \\ \frac{D^2}{(D-\epsilon)^2} &\leq \frac{n^{\frac{2}{3}} + 2n}{2+2n} \\ D &\leq \frac{w}{f_{\text{rel}}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} = D_0 \end{aligned}$$

Furthermore, note that $D_0 < \frac{w}{f_{\text{rel}}} (1 + 2n^{\frac{1}{3}})$ for $n > 2$, hence the hypothesis that $f_{\text{src}} \geq f_{\text{rel}}$ is valid for the values considered. Finally, if the deadline is smaller than the threshold value D_0 , then we can guarantee that the optimal solution will not do any re-execution. However, if the deadline is larger, we do not know what happens (but it can be computed as a function of f_{min} , f_{max} and f_{rel}). \square

Beyond the case analysis itself, the result of Proposition 2 is interesting: we observe that in all cases, the source task is executed faster than the other tasks. This shows that Proposition 1 does not hold for general DAGs, and suggests that some tasks may be more critical than others. A hierarchical approach, that categorizes tasks with different priorities, will guide the design of type B heuristics in Section 5.

4 VDD-HOPPING model

Contrarily to the CONTINUOUS model, the VDD-HOPPING model uses discrete speeds. A processor can choose among a set $\{f_1, \dots, f_m\}$ of possible speeds. A task can be executed at different speeds.

Let $\alpha_{(i,j)}$ be the time of computation of task T_i at speed f_j . The execution time of a task T_i is $\mathcal{E}xe(T_i) = \sum_{j=1}^m \alpha_{(i,j)}$, and the energy consumed during the execution is $E_i = \sum_{j=1}^m \alpha_{(i,j)} f_j^3$. Finally, for the reliability, the approximation used in Equation (2) still holds. However, the reliability of a task is now the product of the reliabilities for each time interval with constant speed, hence $R_i = \prod_{j=1}^m (1 - \lambda_0 e^{-df_j \alpha_{(i,j)}})$. Using a first order approximation, we obtain

$$R_i = 1 - \lambda_0 \sum_{j=1}^m e^{-df_j \alpha_{(i,j)}} = 1 - \lambda_0 \sum_{j=1}^m h_j \alpha_{(i,j)}, \text{ where } h_j = e^{-df_j}, 1 \leq j \leq m. \quad (5)$$

We first show that only two different speeds are needed for the execution of a task. This result was already known for the bi-criteria problem makespan/energy, and it is interesting to see that reliability does not alter it:

Proposition 3. *With the VDD-HOPPING model, each task is computed using at most two different speeds.*

Proof. Suppose that a task is computed with three speeds, $f_1 \leq f_2 \leq f_3$, and let $h_j = e^{-df_j}$, for $j = 1, 2, 3$. We show that we can get rid of one of those speeds. The proof will follow by induction. Let α_i be the time spent by the processor at speed f_i . We aim at replacing each α_i by some α'_i so that we have a better solution. The constraints write:

1. Deadline not exceeded:

$$\alpha_1 + \alpha_2 + \alpha_3 \geq \alpha'_1 + \alpha'_2 + \alpha'_3. \quad (6)$$

2. Same amount of work:

$$\alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 = \alpha'_1 f_1 + \alpha'_2 f_2 + \alpha'_3 f_3. \quad (7)$$

3. Reliability preserved:

$$\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3 \geq \alpha'_1 h_1 + \alpha'_2 h_2 + \alpha'_3 h_3. \quad (8)$$

4. Less energy spent:

$$\alpha_1 f_1^3 + \alpha_2 f_2^3 + \alpha_3 f_3^3 > \alpha'_1 f_1^3 + \alpha'_2 f_2^3 + \alpha'_3 f_3^3. \quad (9)$$

We show that $\alpha'_1 = \alpha_1 - \epsilon_1$, $\alpha'_2 = \alpha_2 + \epsilon_1 + \epsilon_3$, and $\alpha'_3 = \alpha_3 - \epsilon_3$ is a valid solution:

- Equation (6) is satisfied, since $\alpha_1 + \alpha_2 + \alpha_3 = \alpha'_1 + \alpha'_2 + \alpha'_3$.
- Equation (7) gives $\epsilon_1 = \epsilon_3 \left(\frac{f_3 - f_2}{f_2 - f_1} \right)$.
- Next we replace the α'_i and ϵ_i in Equation (8) and we obtain $h_2(f_3 - f_1) \leq h_1(f_3 - f_2) + h_3(f_2 - f_1)$, which is always true by convexity of the exponential (since $h_j = e^{-df_j}$).
- Finally, Equation (9) gives us $\epsilon_1 f_1^3 + \epsilon_3 f_3^3 > (\epsilon_3 + \epsilon_1) f_2^3$, which is necessarily true since $f_1 < f_2 < f_3$ and $f \rightarrow f^3$ is convex (barycenter).

Since we want all the α'_i to be nonnegative, we take

$$\epsilon_1 = \min \left(\alpha_1, \alpha_3 \left(\frac{f_3 - f_2}{f_2 - f_1} \right) \right) \quad \text{and} \quad \epsilon_3 = \min \left(\alpha_3, \alpha_1 \left(\frac{f_2 - f_1}{f_3 - f_2} \right) \right).$$

We have either $\epsilon_1 = \alpha_1$ or $\epsilon_3 = \alpha_3$, which means that $\alpha'_1 = 0$ or $\alpha'_3 = 0$, and we can indeed compute the task with only two speeds, meeting the constraints and with a smaller energy. \square

We are now ready to assess the problem complexity:

Theorem 3. *The TRI-CRIT-VDD-CHAIN problem is NP-complete.*

The proof is similar to that of Theorem 1, assuming that there are only two available speeds, f_{\min} and f_{\max} . Then we reduce the problem from SUBSET-SUM. Note that here again, the problem turns out to be NP-hard even with one single processor (linear chain of tasks).

Proof. Consider the associated decision problem: given an execution graph, m possible speeds, a deadline, a reliability, and a bound on the energy consumption, can we find the time each task will spend at each speed such that the deadline, the reliability and the bound on energy are respected? The problem is clearly in NP: given the time spent in each speed for each task, computing the execution time, the reliability and the energy consumption can be done in polynomial time. To establish the completeness, we use a reduction from SUBSET-SUM [11]. Let \mathcal{I}_1 be an instance of SUBSET-SUM: given n strictly positive integers a_1, \dots, a_n , and a positive integer X , does there exist a subset I of $\{1, \dots, n\}$ such that $\sum_{i \in I} a_i = X$? Let $S = \sum_{i=1}^n a_i$.

We build the following instance \mathcal{I}_2 of our problem. The execution graph is a linear chain with n tasks, where:

- task T_i has weight $w_i = a_i$;
- the processor can run at $m = 2$ different speeds, f_{\min} and f_{\max} ;
- $\lambda_0 = \frac{f_{\max}}{100 \max_{i=1..n} a_i}$;
- $f_{\min} = \sqrt{\lambda_0} f_{\max} \max_{i=1..n} a_i = \frac{f_{\max}}{10}$;
- $f_{\text{rel}} = f_{\max}$; $d = 0$.

The bounds on reliability, deadline and energy are:

- $R_i^0 = R_i(f_{\text{rel}}) = 1 - \lambda_0 \frac{w_i}{f_{\text{rel}}}$ for $1 \leq i \leq n$;
- $D_0 = \frac{2X}{f_{\min}} + \frac{S-X}{f_{\max}}$;
- $E_0 = 2X f_{\min}^2 + (S-X) f_{\max}^2$.

Clearly, the size of \mathcal{I}_2 is polynomial in the size of \mathcal{I}_1 .

Suppose first that instance \mathcal{I}_1 has a solution, I . For all $i \in I$, T_i is executed twice at speed f_{\min} . Otherwise, for all $i \notin I$, it is executed at speed f_{\max} one time only. The execution time is $\frac{2 \sum_{i \in I} a_i}{f_{\min}} + \frac{\sum_{i \notin I} a_i}{f_{\max}} = \frac{2X}{f_{\min}} + \frac{S-X}{f_{\max}} = D$. The reliability is met for all tasks not in I , since they are executed at speed f_{rel} . It is also met for all tasks in I : $\forall i \in I, 1 - \lambda_0^2 \frac{x_i^2}{f_{\min}^2} \geq 1 - \lambda_0 \frac{w_i}{f_{\max}}$. The energy consumption is $E = \sum_{i \in I} 2a_i f_{\min}^2 + \sum_{i \notin I} a_i f_{\max}^2 = 2X f_{\min}^2 + (S - X) f_{\max}^2 = E_0$. All bounds are respected, and therefore the execution speeds are a solution to \mathcal{I}_2 (and each task keeps a constant speed during its whole execution).

Suppose now that \mathcal{I}_2 has a solution. Since we consider the VDD-HOPPING model, each execution can be run partly at speed f_{\min} , and partly at speed f_{\max} . However, tasks executed only once are necessarily executed only at maximum speed to match the reliability constraint.

Let $I = \{i \mid T_i \text{ is executed twice in the solution}\}$. Let $Y = \sum_{i \in I} a_i$. We have $2Y = Y_1 + Y_2$, where Y_1 is the total weight of each execution and re-execution ($2Y$) of tasks in I that are executed at speed f_{\min} , and Y_2 the total weight that is executed at speed f_{\max} . We show that necessarily $Y_1 = 2X = 2Y$, i.e., no part of any task in I is executed at speed f_{\max} .

First let us show that $2X \leq 2Y$. The energy consumption of the solution of \mathcal{I}_2 is $E = Y_1 f_{\min}^2 + Y_2 f_{\max}^2 + (S - Y) f_{\max}^2 = Y_1 f_{\min}^2 + (S - Y_1 + Y) f_{\max}^2$. By differentiating this function (with regards to Y_1 , $E' = f_{\min}^2 - f_{\max}^2 < 0$), we can see that the minimum is reached for $Y_1 = 2Y$ (since $Y_1 \in [0, 2Y]$). Then, for $Y_1 = 2Y$, since the solution is such that $E \leq E_0$, we have $E - E_0 = (Y - X)(2f_{\min}^2 - f_{\max}^2) \leq 0$, and therefore $X \leq Y$.

Next let us show that $Y_1 \leq 2X$. Suppose by contradiction that $Y_1 > 2X$, then the execution time of the solution of \mathcal{I}_2 is $D = \frac{Y_1}{f_{\min}} + \frac{Y_2}{f_{\max}} + \frac{S-Y}{f_{\max}} = \frac{Y_1}{f_{\min}} + \frac{S-Y_1+Y}{f_{\max}}$. By differentiating this function (with regards to Y_1), we can see it is strictly increasing when Y_1 goes from $2X$ to $2Y$. However, when $Y_1 = 2X + \epsilon$, $D - D_0 = \frac{\epsilon}{f_{\min}} + \frac{Y-X+\epsilon}{f_{\max}} > 0$ (indeed, each value of the sum is strictly positive). Hence, $Y_1 \leq 2X$.

Finally, let us show that $Y_1 = 2X = 2Y$. Since \mathcal{I}_2 is a solution, we know that $E \leq E_0$, and therefore $2X - Y_1 \geq (Y + X - Y_1) \frac{f_{\max}^2}{f_{\min}^2} \geq (Y + X - Y_1)$ (the last equality is only met when $Y + X - Y_1 = 0$). Hence $2X \geq X + Y$, which is only possible if $2X = X + Y$. This gives us the final result: $Y_1 = 2X = 2Y$ (all inequalities are tight).

We conclude that $\sum_{i \in I} a_i = X$, and therefore \mathcal{I}_1 has a solution. This concludes the proof. \square

In the following, we propose some polynomial time heuristics to tackle the general tri-criteria problem. While these heuristics are designed for the CONTINUOUS model, they can be easily adapted to the VDD-HOPPING model thanks to Proposition 3.

5 Heuristics for TRI-CRIT-CONT

In this section, building upon the theoretical results of Section 3, we propose some polynomial time heuristics for the TRI-CRIT-CONT problem, which was shown NP-hard (see Theorem 1). Recall that the mapping of the tasks onto the processors is given, and we aim at reducing the energy consumption by exploiting re-execution and speed scaling, while meeting the deadline bound and all reliability constraints.

The first idea is inspired by Proposition 1: first we search for the optimal solution of the problem instance without re-execution, a phase that we call *deceleration*: we slow down some tasks if it can save energy without violating one of the constraints. Then we refine the schedule and choose the tasks that we want to re-execute, according to some criteria. We call *type A heuristics* such heuristics that obey this general scheme: first deceleration then re-execution. Type A heuristics are expected to be efficient on a DAG with a low degree of parallelism (optimal for a chain).

However, Proposition 2 (with fork graphs) shows that it might be better to re-execute highly parallel tasks before decelerating. Therefore we introduce *type B heuristics*, which first choose the set of tasks to be re-executed, and then try to slow down the tasks that could not be re-executed. We need to find good criteria to select which tasks to re-execute, so that type B heuristics prove efficient for DAGs with a high degree of parallelism. In summary, type B heuristics obey the opposite scheme: first re-execution then deceleration.

For both heuristic types, the approach for each phase can be sketched as follows. Initially, each task is executed once at speed f_{\max} . Then, let d_i be the finish time of task T_i in the current configuration:

- *Deceleration*: We select a set of tasks that we execute at speed $f_{\text{dec}} = \max(f_{\text{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\text{max}})$, which is the slowest possible speed meeting both the reliability and deadline constraints.
- *Re-execution*: We greedily select tasks for re-execution. The selection criterion is either by decreasing weights w_i , or by decreasing *super-weights* W_i . The super-weight of a task T_i is defined as the sum of the weights of the tasks (including T_i) whose execution interval is included into T_i 's execution interval. The rationale is that the super-weight of a task that we slow down is an estimation of the total amount of work that can be slowed down together with that task, hence of the energy potentially saved: this corresponds to the total slack that can be reclaimed.

We introduce further notations before listing the heuristics:

- *SUS* (Slack-Usage-Sort) is a function that sorts tasks by decreasing super-weights.
- *ReExec* is a function that tries to re-execute the current task T_i , at speed $f_{\text{re-ex}} = \frac{2c}{1+c} f_{\text{rel}}$, where $c = 4\sqrt{\frac{2}{7}} \cos \frac{1}{3}(\pi - \tan^{-1} \frac{1}{\sqrt{7}}) - 1$ (≈ 0.2838) (note that $f_{\text{re-ex}}$ is the optimal speed in the proof of Theorem 1). If it succeeds, it also re-executes at speed $f_{\text{re-ex}}$ all the tasks that are taken into account to compute the super-weight of T_i . Otherwise, it does nothing.
- *ReExec&SlowDown* performs the same re-executions as *ReExec* when it succeeds. But if the re-execution of the current task T_i is not possible, it slows down T_i as much as possible and does the same for all the tasks that are taken into account to compute the super-weight of T_i .

We now detail the heuristics:

Hf_{max}. In this heuristic, tasks are simply executed once at maximum speed.

Hno-reex. In this heuristic, we do not allow any re-execution, and we simply consider the possible deceleration of the tasks. We set a uniform speed for all tasks, equal to f_{dec} , so that both the reliability and deadline constraints are matched. Note that heuristics **Hf_{max}** and **Hno-reex** are identical except for a constant ratio on the speeds of each task, $\frac{f_{\text{max}}}{f_{\text{dec}}}$. Therefore, the energy ratio $\frac{E_{\text{Hf}_{\text{max}}}}{E_{\text{Hno-reex}}}$ is always equal to $\left(\frac{f_{\text{max}}}{f_{\text{dec}}}\right)^2$ (for instance, if $f_{\text{max}} = 1$ and $f_{\text{dec}} = 2/3$, then the energy ratio is equal to 2.25).

A.Greedy. This is a type A heuristic, where we first set the speed of each task to f_{dec} (deceleration). Let Greedy-List be the list of all the tasks sorted according to decreasing weights w_i . Each task T_i in Greedy-List is re-executed at speed $f_{\text{re-ex}}$ whenever possible. Finally, if there remains some slack at the end of the processing, we slow down both executions of each re-executed task as much as possible.

A.SUS-Crit. This is a type A heuristic, where we first set the speed of each task to f_{dec} . Let List-SW be the list of all tasks that belong to a critical path, sorted according to SUS. We apply ReExec to List-SW (re-execution). Finally we reclaim slack for re-executed tasks, similarly to the final step of **A.Greedy**.

B.Greedy. This is a type B heuristic. We use Greedy-List as in heuristic **A.Greedy**. We try to re-execute each task T_i of Greedy-List when possible. Then, we slow down both executions of each re-executed task T_i of Greedy-List as much as possible. Finally, we slow down the speed of each task of Greedy-List that turn out not re-executed, as much as possible.

B.SUS-Crit. This is a type B heuristic. We use List-SW as in heuristic **A.SUS-Crit**. We apply ReExec to List-SW (re-execution). Then we run Heuristic B.Greedy.

B.SUS-Crit-Slow. This is a type B heuristic. We use List-SW, and we apply ReExec&SlowDown (re-execution). Then we use Greedy-List: for each task T_i of Greedy-List, if there is enough time, we execute twice T_i at speed $f_{\text{re-ex}}$ (re-execution); otherwise, we execute T_i only once, at the slowest admissible speed.

Best. This is simply the minimum value over the seven previous heuristics, for reference.

The complexity of all these heuristics is bounded by $O(n^4 \log n)$, where n is the number of tasks. The most time-consuming operation is the computation of List-SW (the list of all elements belonging to a critical path, sorted according to SUS).

6 Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 5. The heuristics were coded in OCaml. The source code is publicly available at [2] (together with additional results that were omitted due to lack of space).

6.1 Simulation settings

In order to evaluate the heuristics, we have generated DAGs using the random DAG generation library GGEN [9]. Since GGEN does not assign a weight to the tasks of the DAGs, we use a function that gives a random float value in the interval $[0, 10]$. Each simulation uses a DAG with 100 nodes and 300 edges. We observe similar patterns for other numbers of edges, see [2] for further information.

We apply a critical-path list scheduling algorithm to map the DAG onto the p processors: we assign the most urgent ready task (with largest bottom-level) to the first available processor. The bottom-level is defined as $bl(T_i) = w_i$ if T_i has no successor task, and $bl(T_i) = w_i + \max_{(T_i, T_j) \in \mathcal{E}} bl(T_j)$ otherwise.

We choose a reliability constant $\lambda_0 = 10^{-5}$ [1] (we obtain identical results with other values, see below). Each reported result is the average on ten different DAGs with the same number of nodes and edges, and the energy consumption is normalized with the energy consumption returned by the **Hno-reex** heuristic. If the value is lower than 1, it means that we have been able to save energy thanks to re-execution.

We analyze the influence of three different parameters: the tightness of the deadline D , the number of processors p , and the reliability speed f_{rel} . In fact, the absolute deadline D is irrelevant, and we rather consider the *deadline ratio* $DEADLINERATIO = \frac{D}{D_{min}}$, where D_{min} is the execution time when executing each task once and at maximum speed f_{max} (heuristic Hf_{max}). Intuitively, when the deadline ratio is close to 1, there is almost no flexibility and it is difficult to re-execute tasks, while when the deadline ratio is larger we expect to be able to slow down and re-execute many tasks, thereby saving much more energy.

6.2 Simulation results

First note that with a single processor, heuristics A.SUS-Crit and A.Greedy are identical, and heuristics B.SUS-Crit and B.Greedy are identical (by definition, the only critical path is the whole set of tasks).

Deadline ratio. In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and $f_{rel} = \frac{2}{3}f_{max}$. Figure 1 reports results for $p = 1$ and $p = 50$. When $p = 1$, we see that the results are identical for all heuristics of type A, and identical for all heuristics of type B. As expected from Proposition 1, type A heuristics are better (see Figure 1a). With more processors (10, 50, 70), the results have the same general shape: see Figure 1b with 50 processors. When DEADLINERATIO is small, type B heuristics are better. When DEADLINERATIO increases up to 1.5, type A heuristics are closer to type B ones. Finally, when DEADLINERATIO gets larger than 5, all heuristics converge towards the same result, where all tasks are re-executed.

Number of processors. In this set of simulations, we let $DEADLINERATIO \in \{1.2, 1.6, 2, 2.4\}$ and $f_{rel} = \frac{2}{3}f_{max}$. Figure 2 confirms that type A heuristics are particularly efficient when the number of processors is small, whereas type B heuristics are at their best when the number of processors is large. Figure 2a confirms the superiority of type B heuristics for tight deadlines, as was observed in Figure 1b.

Reliability f_{rel} . In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and $DEADLINERATIO \in \{1, 1.5, 3\}$. In Figure 3, there are four different curves: the line at 1 corresponds to Hno-reex and Hf_{max} , then come the heuristics of type A (that all obtain exactly the same results), then B.SUS-Crit and B.Greedy that also obtain the same results, and finally the best heuristic is B.SUS-Crit-Slow. Note that B.SUS-Crit and B.Greedy return the same results because they have the same behavior when $DEADLINERATIO = 1$: there is no liberty of action on the critical paths. However B.SUS-Crit-Slow gives better results because of the way it decelerates the important tasks that cannot be re-executed. When DEADLINERATIO is really tight (equal to 1), decreasing the value of f_{rel} from 1 to 0.9 makes a real difference with type B heuristics. We observe an energy gain of 10% when the number of processors is small (10 in Figure 3a) and of 20% with more processors (50 in Figure 3b).

Reliability constant λ_0 . In Figure 4, we let λ_0 vary from 10^{-5} to 10^{-6} , and observe very similar results throughout this range of values. Note that we did not plot Hf_{max} in this figure to ease the readability.

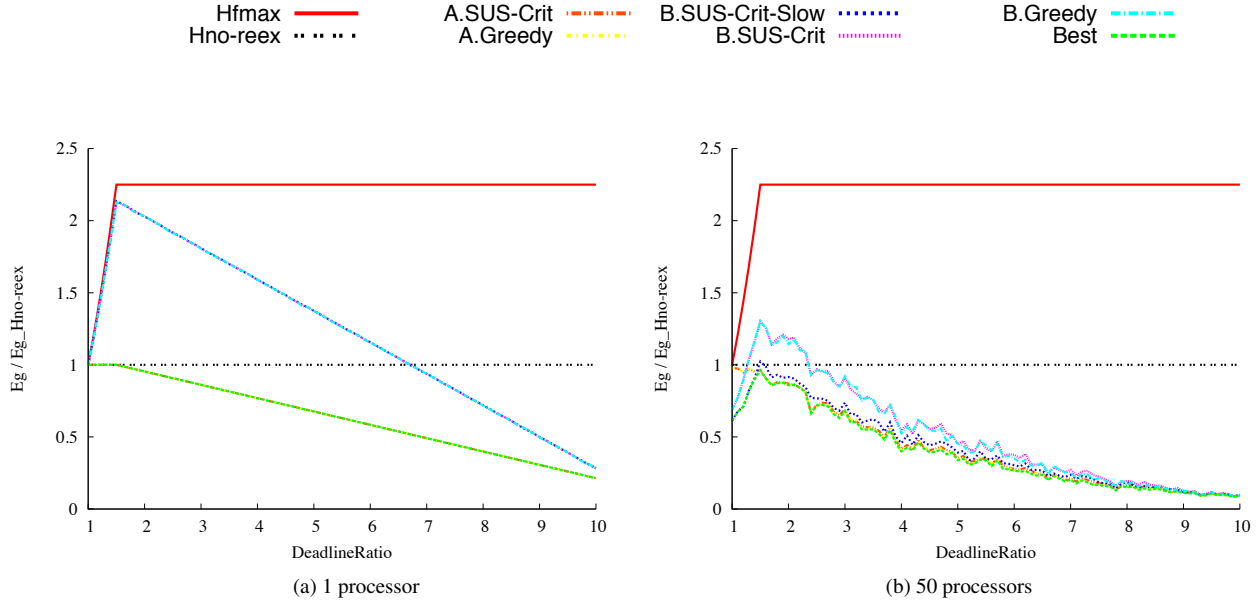


Figure 1: Comparative study when the deadline ratio varies.

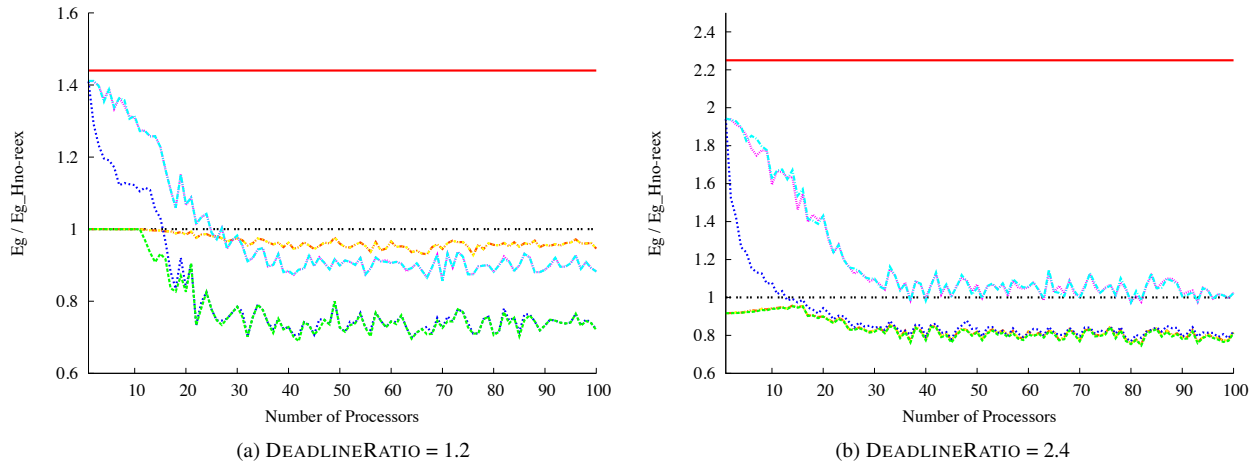


Figure 2: Comparative study when the number of processors p varies.

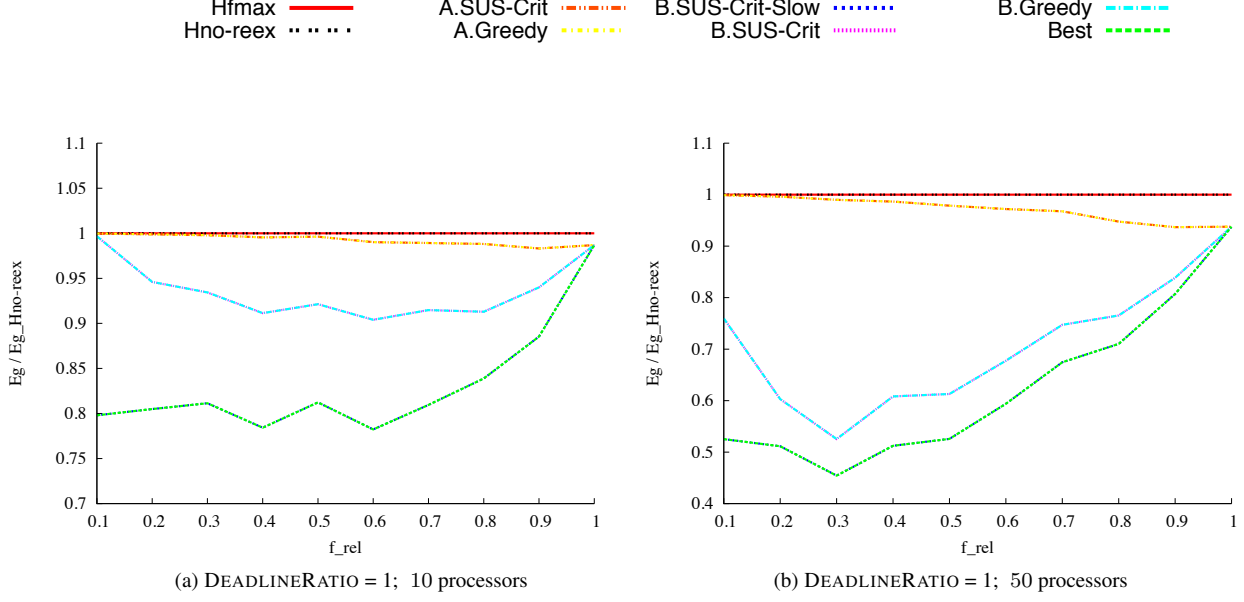


Figure 3: Comparative study when the reliability f_{rel} varies.

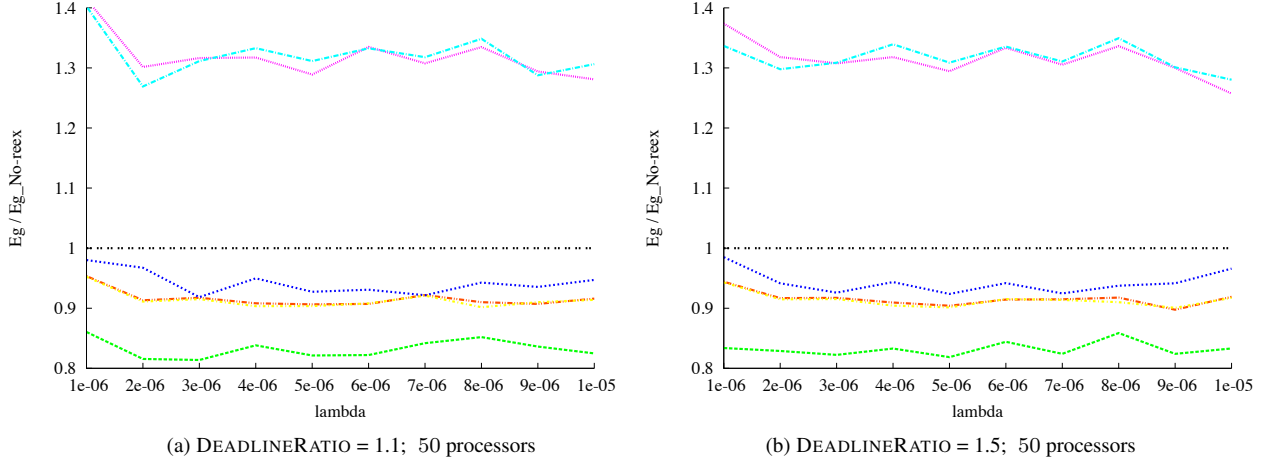


Figure 4: Comparative study when λ_0 varies.

6.3 Understanding the results

A.SUS-Crit and A.Greedy, and B.SUS-Crit and B.Greedy, often obtain similar results, which might lead us to underestimate the importance of critical path tasks. However, the difference between B.SUS-Crit-Slow and B.SUS-Crit shows otherwise. Tasks that belong to a critical path must be dealt with first.

A striking result is the impact of both the number of processors and the deadline ratio on the effectiveness of the heuristics. Heuristics of type A, as suggested by Proposition 1, have much better results when there is a small number of processors. When the number of processors increases, there is a difference between small and large deadline ratio. In particular, when the deadline ratio is small, heuristics of type B have better results. Indeed, heuristics of type A try to accommodate as many tasks as possible, and as a consequence, no task can be re-executed. On the contrary,

heuristics of type B try to favor some tasks that are considered as important. This is highly profitable when the deadline is tight.

Note that all these heuristics take in average less than one *ms* to execute on one instance, which is very reasonable. The heuristics that compute the critical path (*SUS-Crit-*) are the longest, and may take up to two seconds when there are few processors. Indeed, the less processors, the more edges there are in the dependence graph once the task graph is mapped, and hence it increases the complexity of finding the critical path. However, with more than ten processors, the running time never exceeds two *ms*.

Altogether we have identified two very efficient and complementary heuristics, A.SUS-Crit and B.SUS-Crit-Slow. Taking the best result out of those two heuristics always gives the best result over all simulations.

7 Conclusion

In this paper, we have accounted for the energy cost associated to task re-execution in a more realistic and accurate way than the best-case model used in [24]. Coupling this energy model with the classical reliability model used in [21], we have been able to formulate a tri-criteria optimization problem: how to minimize the energy consumed given a deadline bound and a reliability constraint? The “antagonistic” relation between speed and reliability renders this tri-criteria problem much more challenging than the standard bi-criteria (makespan, energy) version. We have stated two variants of the problem, for processor speeds obeying either the CONTINUOUS or the VDD-HOPPING model. We have assessed the intractability of this tri-criteria problem, even in the case of a single processor. In addition, we have provided several complexity results for particular instances.

We have designed and evaluated some polynomial-time heuristics for the TRI-CRIT-CONT problem that are based on the failure probability, the task weights, and the processor speeds. These heuristics aim at minimizing the energy consumption while enforcing reliability and deadline constraints. They rely on *dynamic voltage and frequency scaling* (DVFS) to decrease the energy consumption. But because DVFS lowers the reliability of the system, the heuristics use *re-execution* to compensate for the loss. After running several heuristics on a wide class of problem instances, we have identified two heuristics that are complementary, and that together are able to produce good results on most instances. The good news is that these results bring the first efficient practical solutions to the tri-criteria optimization problem, despite its theoretically challenging nature. In addition, while the heuristics do not modify the mapping of the application, it is possible to couple them with a list scheduling algorithm, as was done in the simulations, in order to solve the more general problem in which the mapping is not already given.

Future work involves several promising directions. On the theoretical side, it would be very interesting to prove a competitive ratio for the heuristic that takes the best out of A.SUS-Crit and B.SUS-Crit-Slow. However, this is quite a challenging work for arbitrary DAGs, and one may try to design approximation algorithms only for special graph structures, e.g., series-parallel graphs. Still, looking back at the complicated case analysis needed for an elementary fork-graph with identical weights (Proposition 2), we cannot underestimate the difficulty of this problem.

While we have designed heuristics for the TRI-CRIT-CONT model in this paper, we could easily adapt them to the TRI-CRIT-VDD model: for a solution given by a heuristic for TRI-CRIT-CONT, if a task should be executed at the continuous speed f , then we would execute it at the two closest discrete speeds that bound f , while matching the execution time and reliability for this task. There remains to quantify the performance loss incurred by the latter constraints.

Finally, we point out that energy reduction and reliability will be even more important objectives with the advent of massively parallel platforms, made of a large number of clusters of multi-cores. More efficient solutions to the tri-criteria optimization problem (makespan, energy, reliability) could be achieved through combining replication with re-execution. A promising (and ambitious) research direction would be to search for the best trade-offs that can be achieved between these techniques that both increase reliability, but whose impact on execution time and energy consumption is very different. We believe that the comprehensive set of theoretical results and simulations given in this paper will provide solid foundations for further studies, and constitute a partial yet important first step for solving the problem at very large scale.

Acknowledgments. A. Benoit and Y. Robert are with the Institut Universitaire de France. This work was supported in part by the ANR *RESCUE* project.

References

- [1] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability power consumption and execution time. In *Proc. of Conf. on Computer Safety, Reliability and Security (SAFECOMP)*, Washington, DC, USA, 2011. IEEE CS Press.
- [2] G. Aupy. Source code and data. <http://gaupy.org/tri-criteria-scheduling>, 2012.
- [3] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert. Reclaiming the energy of a schedule: models and algorithms. *Concurrency and Computation: Practice and Experience*, 2012. Also available as INRIA research report 7598 at gaupy.org. Short version appeared in SPAA'11.
- [4] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 113–121. IEEE CS Press, 2003.
- [5] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *Proc. of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 170–177. ACM Press, 2003.
- [6] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1 – 39, 2007.
- [7] P. Brucker. *Scheduling Algorithms*. Springer, 2007.
- [8] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks. In *Proc. of Int. Conf. on Parallel Processing (ICPP)*, pages 13–20. IEEE CS Press, 2005.
- [9] D. Cordeiro, G. MouniÃ©, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner. Random graph generation for scheduling simulations. In *Proc. of 3rd Int. ICST Conf. on Simulation Tools and Techniques (SIMUTools 2010)*, page 10, mar 2010.
- [10] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Trans. Very Large Scale Integr. Syst.*, 13:1157–1166, October 2005.
- [11] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [12] A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *J. Parallel Distrib. Comput.*, 69:326–336, March 2009.
- [13] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of Annual Design Automation Conf. (DAC)*, pages 806–809, 2000.
- [14] R. Melhem, D. Mosse, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. on Computers*, 53:2004, 2003.
- [15] S. Miermont, P. Vivet, and M. Renaudin. A Power Supply Selector for Energy- and Area-Efficient Local Dynamic Voltage Scaling. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 4644, pages 556–565. Springer Berlin / Heidelberg, 2007.
- [16] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for BlueGene/L systems. In *Proc. of Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 64–73, 2004.
- [17] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of IEEE/ACM Int. Conf. on Hardware/software codesign and system synthesis (CODES+ISSS)*, pages 233–238, 2007.

- [18] R. B. Prathipati. Energy efficient scheduling techniques for real-time embedded systems. Master's thesis, Texas A&M University, May 2004.
- [19] V. J. Rayward-Smith, F. W. Burton, and G. J. Janacek. Scheduling parallel programs assuming preallocation. In P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.
- [20] W. Rudin. *Principles of mathematical analysis*. McGraw-Hill Book Co., New York, third edition, 1976. International Series in Pure and Applied Mathematics.
- [21] S. M. Shatz and J.-P. Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38:16–27, 1989.
- [22] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS. In *Proc. of CCGrid'2010, the 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, pages 368–377, May 2010.
- [23] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. of Conf. on Design, Automation and Test in Europe (DATE)*, page 10918. IEEE CS Press, 2003.
- [24] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 528–534, 2006.
- [25] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 35–40, Washington, DC, USA, 2004. IEEE CS Press.