Energy-aware scheduling under reliability and makespan constraints

Guillaume Aupy, Anne Benoit, Yves Robert

INRIA

Energy-aware scheduling under reliability and makespan constraints

Guillaume Aupy — Anne Benoit — Yves Robert

N° 7757

October 2011

Distributed and High Performance Computing

Rapport de recherche

Energy-aware scheduling under reliability and makespan constraints

Guillaume Aupy , Anne Benoit , Yves Robert

Abstract: We consider a task graph to be executed 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 a model frequently used to reduce the energy consumption of a schedule, but it has negative effect on its reliability. 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) with two different speed models: either processors can have arbitrary speeds (continuous speeds), or a processor can run at a finite number of different speeds, and it can change its speed during a computation. We propose several novel tri-criteria scheduling heuristics under the continuous speed model, and we evaluate them through a set of simulations. Our two best heuristics turn out to be very efficient and complementary.

Key-words: Scheduling; energy; fault-tolerance; reliability; re-execution; models; complexity; polynomial heuristics.

Minimisation de l'énergie d'un ordonnancement sous contraintes de fiabilité et de temps d'exécution

Résumé : Considérons un graphe de tâches devant être exécutées sur un ensemble de processeurs homogènes. Notre but est de minimiser la consommation d'énergie, sans dépasser une limite de temps, et en respectant une borne de fiabilité. Un modèle fréquemment utilisé pour réduire la consommation d'énergie, est le changement dynamique de voltage et de fréquence (DVFS). Malheureusement ses effets sur la fiabilité du systèmes sont négatifs. Dans ce travail, afin de concilier diminution de l'énergie du système et contrainte sur la fiabilité, nous autorisons la ré-exécution de certaines tâches. Nous donnerons la complexité du problème d'ordonnancement tri-critère introduit (énergie, fiabilité, temps d'exécution) pour deux modèles de vitesse différents : d'une part les processeurs peuvent utiliser n'importe quelle vitesse (modèle continu), ou un processeur ne peut travailler que sur un nombre fini de vitesses, mais il peut changer de vitesse pendant son exécution. Nous proposons plusieurs heuristiques polynomiales pour résoudre ces problèmes tri-critères. Puis nous les évaluons au travers d'un ensemble de simulation. Il se trouve que nos deux meilleures heuristiques sont très efficaces et complémentaires.

Mots-clés : Ordonnancement; modèles énergétiques; tolérance aux fautes; fiabilité; ré-exécution; modèles; complexité; heuristiques polynômiales.

Contents

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, 7], 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, 7]. Two popular models for processor speeds are the following:

CONTINUOUS: 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].

VDD-HOPPING: A processor can run at a finite number of different speeds $(f_1, ..., f_m)$. It can also change its speed during a computation (*hopping* between different voltages, and hence speeds). Any rational speed can therefore be simulated [13]. 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.

While energy consumption can be reduced by using speed scaling techniques, it was shown in [24, 8] that speed scaling increases the number of transient fault rates of the system. In order to make up for the loss in reliability due to the energy efficiency, different models have been proposed for fault-tolerance.

- *Re-execution:* this 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 [24, 16].
- *Replication:* this model, studied in [1, 10], consists in executing the same task on p different processors simultaneously, in order to meet the reliability constraints.
- *Checkpointing:* this model, studied in [12, 22], consists in "saving" the work done at some certain points of the work, hence reducing the amount of work lost when a failure occurs.

This work focuses on the re-execution model, for several reasons. On the one hand, replication is too costly in terms of energy consumption: even if the first execution turns out successful (no failure occurred), the other executions will still have to take place. 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.

Consider a Directed Acyclic Graph (DAG) of n tasks that has to be computed on p identical processors. The traditional scheduling objective consists in minimizing the execution time, or makespan, to process the DAG. In order to do so, the DAG is mapped on the processors and an execution speed is assigned to each task of the DAG. Using these speeds, we can define an energy and a reliability for the DAG. In this work, we present theoretical results and novel tri-criteria heuristics that use re-execution in order to minimize the energy consumption under the constraints of both a reliability threshold and a deadline bound. These criteria are formally defined in Section 3.

The rest of the paper is organized as follows. We survey related work in Section 2 before presenting our three main contributions. The first contribution is a formal model of the tri-criteria scheduling problem under consideration (Section 3). The second contribution is to provide theoretical results for the different speed models, CONTINUOUS (Section 4) and VDD-HOPPING (Section 5). The third contribution is the design of tri-criteria scheduling heuristics that use re-execution to increase the reliability of a system under the continuous model (Section 6), and their evaluation through extensive simulations (Section 7). To the best of our knowledge, this is the first attempt to propose practical solutions to this tri-criteria problem. Finally, we give concluding remarks in Section 8.

2   Related work

Since the introduction of DVFS, many papers have dealt with the optimization of energy consumption while enforcing a deadline [4, 6, 7, 3]. In our previous work [3], we consider a task graph to be executed on a set of processors. We assume that the mapping is given, and we aim at optimizing the energy consumption while enforcing a prescribed bound on the execution time. 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 [11, 17]. In [3], we have shown that this problem has polynomial complexity both for the CONTINUOUS and VDD-HOPPING speed models, but it is NP-hard for discrete speeds. In this work, we tackle a much more challenging problem, since we do not assume that the mapping of the graph is given, and we investigate the impact of re-execution to increase reliability.

Even though it has been pointed out that DVFS increases the probability of failures exponentially, and that this probability cannot be neglected in large-scale computing [14], very few authors have tackled the problem of optimizing the three criteria simultaneously: makespan, reliability and energy. The closest works to ours are [16, 24, 1].

Izosinov et al [16] study a tri-criteria optimization problem. They consider heterogeneous architectures. However, the DAG is already mapped on the architecture and each processor has the same set of speeds, so the fact that the architecture is heterogeneous does not really appear. They do not have any formal energy model. Furthermore, and this is the reason why we will not be able to compare our work to their method, they assume that the user will specify the maximum number of failures per processor tolerated to satisfy the reliability constraint.

Zhu et al [24] are also addressing a tri-criteria optimization problem: minimizing the energy consumption while enforcing a deadline and matching reliability constraints. In order to do so, they choose some tasks that will have to be re-executed in order to match the reliability constraint. They simplify the scheduling problem by working on a single processor (as we will see, the problem is still NP-complete).

Finally, Assayad et al [1] have recently proposed an off-line tri-criteria scheduling heuristic (TSH). TSH uses active replication to minimize the schedule length, its global failure rate and its power consumption. They work on a homogeneous platform, fully connected. 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. Their heuristic however has the advantage of taking communication costs between processors into account. Future work will be devoted to compare our heuristics to TSH.

3   Models

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where $V = \{T_1, T_2, \ldots, T_n\}$ 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 assume that we have 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 $\{f_1, \cdots, f_m\}$), depending on the speed model. The goal is to the 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 will be executed once, and some tasks will be re-executed. We detail below the conditions that are enforced on the corresponding execution speeds.

In this section, for the sake of clarity, we assume that a task is executed at the same (unique) speed throughout execution. In Section 4, we show that this strategy is optimal for the CONTINUOUS model; in Section 5, we show that only two different speeds are needed for the VDD-HOPPING model (and we update the corresponding formulas accordingly).

3.1 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. A schedule specifies the tasks that are re-executed, and the speed at which each task is executed (and possibly re-executed), and its makespan should not be greater than D.

Let $\mathcal{E}xe(w_i, f_i)$ be the execution time of a task T_i of weight w_i at speed f_i. We assume that the cache size is adapted to the application, therefore ensuring that the execution time is linearly related to the frequency [12]: $\mathcal{E}xe(w_i, f_i) = \frac{w_i}{f_i}$. When a task is scheduled to be re-executed at two different speeds f_i^1 and f_i^2, we always account for both executions, even when the first execution is successful: 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.

3.2 Reliability

We use the fault model of Zhu and Aydin [23]. *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:

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

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_i is:

$$R_i(f_i) = e^{-\lambda(f_i) \times \mathcal{E}xe(w_i, f_i)}.$$

Because the fault rate is usually very small, of the order of 10^{-6} per time unit in [5, 16], 10^{-5} in [1], we can use the first order approximation of $R_i(f_i)$ as

$$\begin{aligned}
R_i(f_i) &= 1 - \lambda(f_i) \times \mathcal{E}xe(w_i, f_i) \\
&= 1 - \tilde{\lambda}_0 \, e^{\tilde{d} \frac{f_{\max} - f_i}{f_{\max} - f_{\min}}} \times \frac{w_i}{f_i} \\
&= 1 - \lambda_0 \, e^{-df_i} \times \frac{w_i}{f_i}, \tag{2}
\end{aligned}$$

where $d = \frac{\tilde{d}}{f_{\max} - f_{\min}}$ and $\lambda_0 = \tilde{\lambda}_0 e^{df_{\max}}$.

We want the reliability of each task T_i to be greater than a given threshold, namely $R_i(f_{\mathtt{rel}})$. If task T_i is executed only once, then $f_{\mathtt{rel}}$ is the minimum speed at which T_i must be executed to match the reliability constraint (recall that the reliability of a task increases with its speed). If task T_i is re-executed, then let $f_i^{(1)}$ be the speed of the first execution and $f_i^{(2)}$ be the speed of the second execution. The execution of T_i is successful if and only if both attempts do not fail, so that the reliability of T_i with re-execution is $R_i = 1 - (1 - R_i(f_i^{(1)}))(1 - R_i(f_i^{(2)}))$, and we want this quantity to be at least equal to $R_i(f_{\mathtt{rel}})$. If task T_i is executed only once at speed f_i, we let $R_i = R_i(f_i)$, and the reliability constraint finally writes:

$$\text{RELIABILITY}: \quad R_i \geq R_i(f_{\mathtt{rel}}) \text{ for } 1 \leq i \leq n . \tag{3}$$

3.3 Energy

If task T_i is executed once at speed f_i , the consumed energy is

$$E_i = E_i(f_i) = \mathcal{E}xe(w_i, f_i) \times f_i^3, \tag{4}$$

which corresponds to the dynamic part of the classical energy models of the literature [4, 6, 7, 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 re-executed at speeds $f_i^{(1)}$ and $f_i^{(2)}$, 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:

$$E_i = E_i(f_i^{(1)}) + E_i(f_i^{(2)}) \ . \tag{5}$$

Note that some authors [23] 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. The total energy consumed by the schedule is

$$\textsc{Energy}: \quad E = \sum_{i=1}^{n} E_i \ , \tag{6}$$

where E_i is defined by Equation (4) if T_i is not re-executed, and by Equation (5) otherwise.

3.4 Optimization problem

With two speed models, we have two variants of the tri-criteria optimization problem.

Definition 1. TRI-CRIT-CONT. Given an application graph $\mathcal{G} = (V, \mathcal{E})$ and p homogeneous processors with continuous speeds, TRI-CRIT-CONT is the problem of minimizing the energy consumption in Equation (6), subject to the deadline bound D and to the reliability constraint of Equation (3).

Definition 2. TRI-CRIT-VDD. This is the same problem as TRI-CRIT-CONT, but with the VDD-HOPPING speed model.

4 Continuous model

If we assume that $f_{\min} = f_{\texttt{rel}} = f_{\max}$, we can easily reduce TRI-CRIT-CONT to a classical scheduling problem that is NP-complete as soon as there are two processors [9]. However this result is not very interesting. We want to show that even without considering the mapping issue, the problem is still NP-hard. In order to do so, in this section we consider *linear chains of tasks*. For a linear chain, we have $\mathcal{E} = \cup_{i=1}^{n-1}\{T_i \rightarrow T_{i+1}\}$, and any list scheduling mapping is always optimal, for any processor number (there is always only one order to map the tasks).

In this section, after establishing the optimality of unique-speed execution per task, we show that finding the solution of TRI-CRIT-CONT on a linear chain and with a single processor is NP-hard. We conclude with additional results that will guide the design of the heuristics in Section 6.

4.1 Optimality of unique-speed execution per task

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

Proof. First let us assume that the function that gives the speed of the execution of a task is a piecewise-constant function. The general proof 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 [19]. Therefore, this proof is valid for any piecewise-continuous function.

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 $W/(t'' - t)$, the same amount of work is done within the same time, and the energy consumption during this interval of time becomes $E' = (t'' - t) \times (W/(t'' - t))^3$. 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, $h(t, t', t'', f, f') < h(t, t', t'', f_d, f_d)$, 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 (when it occurs) is executed at the same speed as the first execution:

Lemma 2. *With the* TRI-CRIT-CONT *model, it is optimal to re-execute each task (whenever needed) at the same speed as its first execution.*

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 $1 - \lambda_0^2 w_i^2 \frac{1}{f^2} = 1 - \lambda_0^2 w_i^2 \frac{1}{f_i f'_i}$. The fact that the deadline constraint is matched is due to the fact that $\sqrt{f_i f'_i} \geq \frac{2 f_i f'_i}{f_i + f'_i}$ (by squaring both sides of the equation we obtain $(f_i - f'_i)^2 \geq 0$). Then we use the fact that $f_d = \frac{2 f_i f'_i}{f_i + f'_i}$ is the minimal speed such that $\forall f \geq f_d$, $\frac{2 w_i}{f} < \frac{w_i}{f_i} + \frac{w_i}{f'_i}$. Finally, it is easy to see that the energy consumption is improved since $2 f_i f'_i \leq f_i^2 + f'^2_i$, hence $2 w_i f_i f'_i \leq w_i f_i^2 + w_i f'^2_i$.

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 $f_2 = \frac{2 f_i f'_i}{f_i + f'_i}$ (minimum speed to match the deadline). Then the fact that the energy does not increase comes from the convexity of this function. $\square$

Note that the unique-speed result applies to any solution of the problem, not just optimal solutions, hence all heuristics of Section 6 will assign a unique speed to each task, be it re-executed or not.

4.2 Intractability of Tri-Crit-Cont

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

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 [9]. Let $\mathcal{I}_1$ be an instance of SUBSET-SUM: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, does there exist a subset I of $\{1, \ldots, n\}$ 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 $w_i = a_i$;
- $\lambda_0 = \frac{f_{\max}}{100 \max_i a_i}$;
- $f_{\min} = \sqrt{\lambda_0 \max_i a_i f_{\max}} = \frac{1}{10} f_{\max}$;
- $D_0 = \frac{S}{f_{\max}} + \frac{X}{c f_{\max}}$, where

  $c = 4\sqrt{\frac{2}{7}} \cos \frac{1}{3}(\pi - \tan^{-1} \frac{1}{\sqrt{7}}) - 1$ (≈ 0.2838);

- $f_{\mathtt{rel}} = f_{\max}$; $d = 0$; $R_i^0 = R_i^{f_{\mathtt{rel}}} = 1 - \lambda_0 \frac{w_i}{f_{\mathtt{rel}}}$;
- $E_0 = 2X(\frac{2c}{1+c} f_{\mathtt{rel}})^2 + (S - X) f_{\mathtt{rel}}^2$.

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 $\frac{2c}{1+c} f_{\mathtt{rel}}$. Otherwise, for all $i \notin I$, it is executed at speed $f_{\mathtt{rel}}$ one time only. The execution time is $\sum_{i \notin I} \frac{a_i}{f_{\mathtt{rel}}} + \sum_{i \in I} 2\frac{a_i}{\frac{2c}{1+c} f_{\mathtt{rel}}} = \frac{S-X}{f_{\mathtt{rel}}} + 2X\frac{1+c}{2c f_{\mathtt{rel}}} = D_0$. The reliability constraint is obviously met for tasks not in I. It is also met for all tasks in I, since $\frac{2c}{1+c} f_{\mathtt{rel}} > f_{\min}$ and two executions at $f_{\min}$ suffice to match the reliability constraint. Indeed, $1 - \lambda_0^2 \frac{a_i^2}{f_{\min}^2} = 1 - \lambda_0 \frac{a_i}{f_{\mathtt{rel}}} \cdot \frac{a_i^2}{\max_i a_i} \geq 1 - \lambda_0 \frac{a_i}{f_{\mathtt{rel}}} = R_i^0$. The energy consumption is exactly E_0. All bounds are respected, and therefore the execution speeds are a solution to $\mathcal{I}_2$.

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

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 Corollary 2, for all $i \in I$, $f_i = f_i'$ (where f' is the speed of the second execution). Because the deadline is the sonly constraint, either $Y = 0$ (no tasks are re-executed), or it is optimal to match the deadline (otherwise we could just slow down one of 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 $E = 2Yf^2 + (S - Y)f_{\mathtt{rel}}^2$ with the constraint $(S - Y)/f_{\mathtt{rel}} + 2Y/f \leq D_0$. First, denote that if $Y = 0$ then $E > E_0$. Hence, we can assume that $Y > 0$. Then we have $f = \frac{2Y}{D_0 f_{\mathtt{rel}} - (S-Y)} f_{\mathtt{rel}}$ because of the previous remark (tight deadline). Plugging back the value of f in E, we have:

$$E(Y) = \left(\frac{(2Y)^3}{(D_0 f_{\mathtt{rel}} - (S-Y))^2} + (S - Y) \right) f_{\mathtt{rel}}^2,$$

which admits a unique minimum when $Y = c(D_0 f_{\mathtt{rel}} - S) = X$. To see this, define $\tilde{Y} = \frac{Y}{D_0 f_{\mathtt{rel}} - S}$. Then we have: $E(\tilde{Y}) = \left(\frac{(2\tilde{Y})^3}{(1+\tilde{Y})^2} + (\frac{S}{D_0 f_{\mathtt{rel}} - S} - \tilde{Y}) \right) (D_0 f_{\mathtt{rel}} - S) f_{\mathtt{rel}}^2$. Differentiating, we ontain $E'(\tilde{Y}) = \left(\frac{3 \cdot 2^3 \tilde{Y}^2}{(1+\tilde{Y})^2} - \frac{2^4 \tilde{Y}^3}{(1+\tilde{Y})^3} - 1 \right) (D_0 f_{\mathtt{rel}} - S) f_{\mathtt{rel}}^2$. Then $E' = 0$ iff

$$24\tilde{Y}^2(1 + \tilde{Y}) - 16\tilde{Y}^3 - (1 + \tilde{Y})^3 = 0. \tag{7}$$

The only positive solution of Equation (7) is $\tilde{Y} = c$, hence the result. Note that when $Y = X$, then $E = E_0$, so any other value of Y will not be valid. There remains to check that the solution matches both constraints on $f_{\min}$ and on reliability. We have $f = \frac{2c}{1+c} f_{\text{rel}} > f_{\min}$, which shows that both constraints hold for task in I. Other tasks are executed at speed f_{rel}. Altogether, we have $\sum_{i \in I} a_i = Y = X$, and therefore $\mathcal{I}_1$ has a solution. This concludes the proof. $\qquad\square$

4.3 Additional results

Proposition 1. *Consider a linear chain execution on a single processor, with the* TRI-CRIT-CONT *model. Suppose $f_{\text{rel}} < f_{\max}$. In any optimal solution, either all tasks are executed only once, and 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 those 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 T_i) and f_j' (new speed of T_j) such that $f_i < f_i' \leq f_{\text{rel}} \leq f_j' < f_j$, 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 T_i is increased while the speed of T_j remains above the reliability threshold.

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

$$w_i f_i^2 + w_j f_j^2 > \qquad w_i(f_i + \epsilon')^2 + w_j(f_j - \epsilon)^2;$$

$$D = \qquad \frac{w_i}{f_i} + \frac{w_j}{f_j} = \frac{w_i}{f_i + \epsilon'} + \frac{w_j}{f_j - \epsilon};$$

$$f_i < \qquad f_i + \epsilon' \leq f_{\text{rel}} \leq f_j - \epsilon < f_j.$$

This is equivalent to:

$$w_i f_i^2 + w_j f_j^2 > \qquad w_i(f_i + \epsilon')^2 + w_j(f_j - \epsilon)^2;$$

$$\epsilon' = \qquad \frac{w_i}{D - \frac{w_j}{f_j - \epsilon}} - f_i;$$

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

Let $\phi : \epsilon \mapsto w_i f_i^2 + w_j f_j^2 - \left(w_i(f_i + \epsilon')^2 + w_j(f_j - \epsilon)^2\right)$. Then $\phi(\epsilon) = \frac{w_i^3 f_j^2}{(Df_j - w_j)^2} - \frac{w_i^3(f_j - \epsilon)^2}{(D(f_j - \epsilon) - w_j)^2} + w_j f_j^2 - w_j(f_j - \epsilon)^2$. We want to prove that ϕ is positive when $0 < \epsilon \leq f_j - \max\left(f_{\text{rel}}, \frac{w_i + w_j}{\frac{w_i}{f_i} + \frac{w_j}{f_j}}\right)$.

Differentiating, we obtain $\phi'(\epsilon) =$

$$\frac{2w_i^3(f_j - \epsilon)}{(D(f_j - \epsilon) - w_j)^2} - \frac{2Dw_i^3(f_j - \epsilon)^2}{(D(f_j - \epsilon) - w_j)^3} + 2w_j(f_j - \epsilon),$$

which simplifies into the polynomial

$$X^3 + 3X^2 + 3X + \frac{w_i^3}{w_j^3} + 1 = 0,$$

by multiplying each side of the equation by $\frac{(D(f_j - \epsilon) - w_j)^3}{w_j^3(f_j - \epsilon)}$, and defining $X = \frac{D(f_j - \epsilon) - w_j}{w_j}$. The only real solution to this polynomial is $-\frac{w_i}{w_j} - 1 < 0$, hence $\forall \epsilon > 0$, $\phi'(\epsilon) < 0$. We deduce that ϕ is minimal when ϵ is maximal, that is: $\epsilon = f_j - \max\left(f_{\text{rel}}, \frac{w_i + w_j}{\frac{w_i}{f_i} + \frac{w_j}{f_j}}\right)$.

Altogether we have found f_i' and f_j' that improve the energy consumption of the schedule. However it was supposed to be optimal, we have a contradiction. $\qquad\square$

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 $f_{\texttt{rel}} > \frac{\sum_{i=1}^n w_i}{D}$, 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 6.

Lemma 3. *With the* TRI-CRIT-CONT *model, when all the task of a DAG have the same weight w, then in any optimal solution, each task is executed (or re-executed) at least at speed f, with:*

$$\lambda_0 w \frac{e^{-2df}}{f} = \frac{e^{-df_{rel}}}{f_{rel}}. \tag{8}$$

Proof. Let us assume first that a task is executed only once at a speed $f_1 < f$. Then, the reliability constraint is not satisfied, and indeed we must have in this case $f_1 \geq f_{\texttt{rel}} > f$.

If the task is re-executed, with one execution at speed f_1 and the other at speed f_2, then if both f_1 and f_2 are strictly smaller than f, the reliability constraint is not satisfied either because of Equation (8). The last case to consider is such that $f_1 < f \leq f_2$, but in this case the solution would not be optimal since it is strictly better to execute the task twice at the same speed, following Lemma 2. $\qquad\square$

In the following, we consider a DAG made of identical tasks, and since a task will never be executed at a speed lower than the speed f defined by Equation (8), we assume that $f_{\min}$ is at least equal to f; otherwise, we can let $f_{\min} = f$, thanks to Lemma 3.

Proposition 2. *Consider a fork graph of $n+1$ identical tasks (a source and $n \geq 2$ independent successors, all of same weight w), executed on $n+1$ processors, with the* TRI-CRIT-CONT *model. In the optimal solution, the source is executed at speed f_{src}, and the n successors are executed at the same speed f_{leaf}. If $D < \frac{2w}{f_{\max}}$, then there is no solution. Otherwise, the number of executions and the values of f_{src} and f_{leaf} depend upon the deadline D as follows:*

1. **No task is re-executed:**
   - if $\frac{2w}{f_{\max}} \leq D \leq \frac{w}{f_{\max}}(1 + n^{\frac{1}{3}})$, then $f_{src} = f_{\max}$ and $f_{leaf} = \frac{w}{Df_{\max} - w} f_{\max}$;
   - if $\frac{w}{f_{\max}}(1 + n^{\frac{1}{3}}) < D \leq \frac{w}{f_{rel}} \frac{1 + n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, then $f_{src} = \frac{w}{D}(1 + n^{\frac{1}{3}})$ and $f_{leaf} = \frac{w}{D} \frac{1 + n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
   - if $\frac{w}{f_{rel}} \frac{1 + n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{rel}}$, then $f_{src} = \frac{w}{Df_{rel} - w} f_{rel}$ and $f_{leaf} = f_{rel}$;
   - if $\frac{2w}{f_{rel}} < D \leq \frac{w}{f_{rel}} \frac{(1 + 2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$, then $f_{src} = f_{leaf} = f_{rel}$.

2. **The source is executed once, and the successors are re-executed:**
   - if $\frac{w}{f_{rel}} \frac{(1 + 2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} < D \leq \frac{w}{f_{rel}}(1 + 2n^{\frac{1}{3}})$, then $f_{src} = \frac{w}{D}(1 + 2n^{\frac{1}{3}})$ and $f_{leaf} = \frac{w}{D} \frac{1 + 2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
   - if $\frac{w}{f_{rel}}(1 + 2n^{\frac{1}{3}}) < D \leq \frac{w}{f_{rel}} 2\sqrt{2}(1 + n^{\frac{1}{3}})$, then $f_{src} = f_{rel}$ and $f_{leaf} = \max(\frac{2w}{Df_{rel} - w} f_{rel}, f_{\min})$.

3. **Each task is re-executed:** *this case is more difficult to characterize. Indeed, it depends on $f_{\min}$, which is dependent on the system and on the weight w of the tasks (see Lemma 3). Hence, even when $D > \frac{w}{f_{rel}} 2\sqrt{2}(1 + n^{\frac{1}{3}})$, the source is not necessarily re-executed, since it may be better to slow down all the successors, if it is possible to run them slow enough. We do not detail these cases.*

Proof. First we recall preliminary results that are valid for any optimal solution of the TRI-CRIT-CONT problem:
   - the re-execution of any task will always be at the same speed than its execution;
   - if a task is executed only once, then $f_{\texttt{rel}} \leq f \leq f_{\max}$;

- if a task is re-executed, then $f_{\min} \leq f < \frac{1}{\sqrt{2}} f_{\mathtt{rel}}$.

Thanks to these preliminary results, we know that 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: when the number of execution is the same, the bijection Energy-Speed is obvious. Because of the intervals of speed depending on the number of execution, we see that for a given energy, if the energy is greater or equal than $w f_{\mathtt{rel}}^2$ then necessarily there is one execution, if it is lower than $w f_{\mathtt{rel}}^2$ then necessarily there are two executions.

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 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 $f_{\mathtt{leaf}}$. The source task may be executed at a different speed, $f_{\mathtt{src}}$.

Next, let us show that the energy consumption of the source is always greater or equal than the one from any leaf in any optimal solution. First, since the source and leaves have the same weight, if we invert the execution speed(s) 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 $n \geq 2$).

We have now fully characterized the shape of any optimal solution. There are only three possibilities:

1. no task is re-executed;
2. the source is executed once and the successors (leaves) are re-executed;
3. each task is re-executed.

Next, we study independently the three cases, i.e., we aim at determining the values of $f_{\mathtt{src}}$ and $f_{\mathtt{leaf}}$ in each case. We will then give conditions on the deadline that indicate what the shape of the solution should be. First we introduce some notations: δ_s is the number of times that the source is executed ($\delta_s = 1$ or $\delta_s = 2$), and $f_{\mathtt{src}}$ is its execution speed(s). Similarly, δ_l is the number of times that the leaves are executed, and $f_{\mathtt{leaf}}$ their execution speeds. Note that with the previous results, $\delta_l \geq \delta_s$. With these notations the problem we want to minimize becomes:

$$
\begin{aligned}
\text{Minimize} \quad & w\delta_s f_{\mathtt{src}}^2 + n \times w\delta_l f_{\mathtt{leaf}}^2 \\
\text{subject to} \quad & \text{(i)} \quad \delta_s \frac{w}{f_{\mathtt{src}}} + \delta_l \frac{w}{f_{\mathtt{leaf}}} \leq D \\
& \text{(ii)} \quad 1 - \lambda_0^{\delta_s} \frac{w^{\delta_s}}{f_{\mathtt{src}}^{\delta_s}} e^{-\delta_s f_{\mathtt{src}}} \geq 1 - \lambda_0 \frac{w}{f_{\mathtt{rel}}} e^{-d f_{\mathtt{rel}}} \\
& \text{(iii)} \quad 1 - \lambda_0^{\delta_l} \frac{w^{\delta_l}}{f_{\mathtt{leaf}}^{\delta_l}} e^{-\delta_l f_{\mathtt{leaf}}} \geq 1 - \lambda_0 \frac{w}{f_{\mathtt{rel}}} e^{-d f_{\mathtt{rel}}}
\end{aligned}
\tag{9}
$$

1. No task is re-executed. Let us assume first that the optimal solution is such that each task is executed only once. From the proof of Theorem 1 in [3], we obtain the optimal speeds with no re-execution; they are given by the following formulas:

- if $D < \frac{2w}{f_{\max}}$ then there is no solution;
- if $\frac{2w}{f_{\max}} \leq D \leq \frac{w}{f_{\max}}(1 + n^{\frac{1}{3}})$, then $f_{\mathtt{src}} = f_{\max}$ and $f_{\mathtt{leaf}} = \frac{w}{D f_{\max} - w} f_{\max}$;
- if $\frac{w}{f_{\max}}(1 + n^{\frac{1}{3}}) < D$, then $f_{\mathtt{src}} = \frac{w}{D}(1 + n^{\frac{1}{3}})$ and $f_{\mathtt{leaf}} = \frac{w}{D} \frac{1 + n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$.

Because there is a minimum speed $f_{\mathtt{rel}}$ to match the reliability, there is a condition when $f_{\mathtt{leaf}} < f_{\mathtt{rel}}$ which makes an amendment on the last item:

- if $\frac{w}{f_{\tt rel}}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D \leq \frac{2w}{f_{\tt rel}}$, then $f_{\tt src} = \frac{w}{Df_{\tt rel}-w}f_{\tt rel}$ and $f_{\tt leaf} = f_{\tt rel}$;
- if $\frac{2w}{f_{\tt rel}} < D$, then $f_{\tt src} = f_{\tt leaf} = f_{\tt rel}$.

2. The source is executed once and the leaves are re-executed. Assume now that all leaves are re-executed. We can consider an equivalent DAG where leaves are of weight $2w$, and a schedule with no re-execution. Then the optimal solution when there is no maximum speed is: $f_{\tt src} = \frac{w}{D}(1 + 2n^{\frac{1}{3}})$ and $f_{\tt leaf} = \frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$. Note that if $f_{\tt leaf} \geq \frac{1}{\sqrt{2}}f_{\tt rel}$, then there is a better solution without re-execution (or there is no solution). Indeed, the solution where the leaves are executed once at speed $\max(\frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\tt rel})$ is a solution, matches the reliability obviously, the deadline ($\frac{w}{\max(\frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\tt rel})} < \frac{2\sqrt{2}w}{f_{\tt rel}} \leq \frac{2w}{\frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}}$), and it has a better energy consumption (because $\frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \geq \frac{1}{\sqrt{2}}f_{\tt rel}$).

Since we are in case 2 with re-execution of the leaves, we can assume that $\frac{w}{f_{\tt rel}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} < D$. Then, depending if $f_{\tt src} \geq f_{\tt rel}$ or $f_{\tt src} < f_{\tt rel}$:

- if $f_{\tt src} \geq f_{\tt rel}$, it means that $D \leq \frac{w}{f_{\tt rel}}(1 + 2n^{\frac{1}{3}})$, ($\frac{w}{f_{\tt rel}}(1 + 2n^{\frac{1}{3}}) > \frac{w}{f_{\tt rel}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$ is true when $n > 2$), then $f_{\tt src} = \frac{w}{D}(1 + 2n^{\frac{1}{3}})$ and $f_{\tt leaf} = \frac{w}{D}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$;
- if $f_{\tt src} < f_{\tt rel}$, it means that $D > \frac{w}{f_{\tt rel}}(1+2n^{\frac{1}{3}})$, then $f_{\tt src} = f_{\tt rel}$ and $f_{\tt leaf} = \max(\frac{2w}{Df_{\tt rel}-w}f_{\tt rel}, f_{\min})$.

3. Each task is re-executed. If the solution is such that each task is re-executed (source or leaf), it is equivalent to consider a DAG with tasks of weight $2w$ and no re-execution. Then the optimal solution when there is no maximum speed is: $f_{\tt src} = \frac{2w}{D}(1 + n^{\frac{1}{3}})$ and $f_{\tt leaf} = \frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$. Note that if $f_{\tt src} \geq \frac{1}{\sqrt{2}}f_{\tt rel}$, then there is a better solution in which the source is not re-executed. Indeed, the solution where the source is executed once at speed $\max(\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\tt rel})$ is a solution, matches the reliability obviously, the deadline ($\frac{w}{\max(\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}, f_{\tt rel})} < \frac{2\sqrt{2}w}{f_{\tt rel}} \leq \frac{2w}{\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}}}$) and has a better energy consumption (because $\frac{2w}{D}\frac{1+n^{\frac{1}{3}}}{n^{\frac{1}{3}}} \geq \frac{1}{\sqrt{2}}f_{\tt rel}$).

Therefore, if $D \leq \frac{w}{f_{\tt rel}}2\sqrt{2}(1 + n^{\frac{1}{3}})$, the source should not be re-executed.

No re-execution $\rightarrow$ leaves re-executed. To complete the proof, there remains to establish the value of the deadline at which re-execution will be used by the optimal solution, i.e., at which point we will move from case 1 to case 2. 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 $D \leq \frac{w}{f_{\tt rel}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}$, then it is better to have no re-execution. Let $D = \frac{w}{f_{\tt rel}}\sqrt{2}\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}} + \epsilon$. Suppose first that $D \leq \frac{w}{f_{\tt rel}}(1 + 2n^{\frac{1}{3}})$, then the energy when the leaves are re-executed is: $E_2 = \frac{w^3}{D^2}(1 + 2n^{\frac{1}{3}})^3$. With no re-execution the total energy is:

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

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

$$2\frac{w^3}{(D-\epsilon)^2}(1+n)\left(\frac{1+2n^{\frac{1}{3}}}{n^{\frac{1}{3}}}\right)^2 \geq \frac{w^3}{D^2}(1+2n^{\frac{1}{3}})^3$$

$$\frac{2}{(D-\epsilon)^2} \frac{1+n}{n^{\frac{2}{3}}} \geq \frac{1+2n^{\frac{1}{3}}}{D^2}$$

$$\frac{D^2}{(D-\epsilon)^2} \geq \frac{n^{\frac{2}{3}}+2n}{2+2n}$$

$$D \geq \frac{w}{f_{\texttt{rel}}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$$

Furthermore, the set $\frac{w}{f_{\texttt{rel}}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}} \leq D \leq \frac{w}{f_{\texttt{rel}}}(1+2n^{\frac{1}{3}})$ is not empty when $n > 2$.

This completes the proof for the boundary between cases 1 and 2: if the deadline is smaller than the threshold value $\frac{w}{f_{\texttt{rel}}} \frac{(1+2n^{\frac{1}{3}})^{\frac{3}{2}}}{\sqrt{1+n}}$, the optimal solution will not do any re-execution. However, if the deadline is larger, then it is better to re-execute the leaves, and hence the optimal solution will be in the shape of case 2.

Source executed once → source re-executed. As we have seen earlier, when $D \leq \frac{w}{f_{\texttt{rel}}} 2\sqrt{2}(1 + n^{\frac{1}{3}})$, the source should not be re-executed, and therefore the optimal solution will be in the shape of case 1 or 2. As stated in the proposition, the cases with larger deadlines are not fully developed here. Hence the proof is complete. $\square$

Beyond the proof 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 6.

5 Vdd-Hopping model

Contrarily to the Continuous model, the Vdd-Hopping model uses discrete speeds. A processor can choose among a set $\{f_1, ..., f_m\}$ of possible speeds. A task can be executed at different speeds. In this section we first show that only two different speeds are needed for the execution of a task under the Vdd-Hopping model. Then we state the intractability of the Tri-Crit-Vdd optimization problem.

5.1 Optimality of two-speed execution per task

In the Vdd-Hopping model, we define $\alpha_{(i,j)}$ as 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 $E(T_i) = \sum_{j=1}^{m} \alpha_{(i,j)} f_j^3$. 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

$$R_i = 1 - \lambda_0 \sum_{j=1}^{m} e^{-df_j} \alpha_{(i,j)} = 1 - \lambda_0 \sum_{j=1}^{m} h_j \alpha_{(i,j)}, \tag{10}$$

where $h_j = e^{-df_j}$ for $1 \leq j \leq m$.

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:
$$\alpha_1 + \alpha_2 + \alpha_3 \geq \alpha_1' + \alpha_2' + \alpha_3'. \tag{11}$$

2. Same amount of work:
$$\alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 = \alpha_1' f_1 + \alpha_2' f_2 + \alpha_3' f_3. \tag{12}$$

3. Reliability preserved:
$$\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3 \geq \alpha_1' h_1 + \alpha_2' h_2 + \alpha_3' h_3. \tag{13}$$

4. Less energy spent:
$$\alpha_1 f_1^3 + \alpha_2 f_2^3 + \alpha_3 f_3^3 > \alpha_1' f_1^3 + \alpha_2' f_2^3 + \alpha_3' f_3^3. \tag{14}$$

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 (11) is satisfied, since $\alpha_1 + \alpha_2 + \alpha_3 = \alpha_1' + \alpha_2' + \alpha_3'$.
- Equation (12) gives $\epsilon_1 = \epsilon_3 \left(\dfrac{f_3 - f_2}{f_2 - f_1} \right)$.
- Next we replace the α_i' and ϵ_i in Equation (13) 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 (14) 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 $f \to f^3$ is convex (barycenter).

Since we want all the α_i' to be nonnegative, we take $\epsilon_1 = \min \left(\alpha_1, \alpha_3 \left(\dfrac{f_3 - f_2}{f_2 - f_1} \right) \right)$ and $\epsilon_3 = \min \left(\alpha_3, \alpha_1 \left(\dfrac{f_2 - f_1}{f_3 - f_2} \right) \right)$.

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. $\square$

5.2 Intractability of Tri-Crit-Vdd

Theorem 2. *The* Tri-Crit-Vdd *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.

Proof. Consider the associated decision problem: given an execution graph, a deadline, m speeds, a reliability, and a bound on the energy consumption, can we find the time each task will spend for each speed such that the deadline, the reliability and the bound on energy are respected? The problem is clearly in NP: given the execution speeds of 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 [9]. Let $\mathcal{I}_1$ be an instance of SUBSET-SUM: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, does there exist a subset I of $\{1, \ldots, n\}$ 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 $f_{\max}$;
- $\lambda_0 = \dfrac{f_{\max}}{100 \max_{i=1..n} a_i}$;
- $f_{\min} = \sqrt{\lambda_0 f_{\max} \max_{i=1..n} a_i} = \dfrac{f_{\max}}{10}$;
- $D_0 = \dfrac{2X}{f_{\min}} + \dfrac{S-X}{f_{\max}}$;
- $f_{\mathtt{rel}} = f_{\max}$; $d = 0$; $R_i^0 = R_i^{f_{\max}} = 1 - \lambda_0 \dfrac{w_i}{f_{\max}}$;

- $E_0 = 2X f_{\min}^2 + (S - X) f_{\max}^2$.

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 $f_{\min}$ (one re-execution also 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. 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 $E = \sum_{i \in I} 2 a_i f_{\min}^2 + \sum_{i \notin I} a_i f_{\max}^2 = 2X f_{\min}^2 + (S - X) f_{\max}^2 = E_0$. All bounds are respected, and therefore the execution speeds are a solution to $\mathcal{I}_2$.

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

Suppose now that $\mathcal{I}_2$ has a solution. In our model, each task can be executed only once or twice. Since we consider the VDD-HOPPING models, each execution can be run partly at speed $f_{\min}$, and partly at speed $f_{\max}$. We first point out that tasks executed only once are necessarily executed only at maximum speed to match the reliability constraint.

Let $I = \{i \mid T_i \text{ is executed twice in the solution }\}$. Note that we have $\sum_{i \notin I} a_i = S - \sum_{i \in I} a_i$.
Let us call $Y = \sum_{i \in I} a_i$, we know that $2Y = Y_1 + Y_2$ 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 $f_{\max}$.
Let us show that necessarily $Y_1 = 2X = 2Y$.
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 it is minimized when $Y_1 = 2Y$ (because $Y_1 \in [0, 2Y]$). Then, when $Y_1 = 2Y$, since we want a solution such that $E \leq E_0$, this is true if and only if $E - E_0 = (Y - X)(2 f_{\min}^2 - f_{\max}^2) \leq 0$. Which gives $X \leq Y$.
Then let us show that $Y_1 \leq 2X$. Suppose at first *by contradiction* that $Y_1 > 2X$, 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, every value of the sum is strictly positive). Hence, $Y_1 \leq 2X$.
Finally let us show that $Y_1 = 2X = 2Y$. Because $\mathcal{I}_2$ is a solution, we know that $E \leq E_0$. Which gives us: $2X - Y_1 \geq (Y + X - Y_1) \frac{f_{\max}^2}{f_{\min}^2} \geq (Y + X - Y_1)$. (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: $Y_1 = 2X = 2Y$ (only case where all equalities are met which was the only possibility).

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

The following result identifies a polynomial instance of the problem:

Theorem 3. *If we consider the TRI-CRIT-VDD model where re-execution is not allowed, the problem for a linear chain of tasks can be solved in polynomial time (via linear programming).*

Proof. As above, for $1 \leq i \leq n$ and $1 \leq j \leq m$, $\alpha_{(i,j)}$ is the time spent at speed f_j for executing task T_i. There are $n \times m$ variables. The constraints are:
- $\sum_{i=1}^{n} \mathcal{E}xe(T_i) = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{(i,j)} \leq D$: the total execution time for all tasks does not exceed the deadline;
- $\forall 1 \leq i \leq n$, $\sum_{j=1}^{m} \alpha_{(i,j)} \times f_j \geq w_i$: task T_i is completely executed;
- $\forall 1 \leq i \leq n$, $\sum_{j=1}^{m} \alpha_{(i,j)} \times h_j \leq \frac{1 - R_i(f_{\text{rel}})}{\lambda_0}$ (see Equation (10));
- $\forall 1 \leq i \leq n$, $\forall 1 \leq j \leq m$, $\alpha_{(i,j)} \geq 0$: all times are positive.

The objective function is then $\min \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{(i,j)} f_j^3 \right)$. The size of this linear program is clearly polynomial in the size of the instance, all nm variables are rational, and therefore it can be solved in polynomial time [20]. $\square$

6 Heuristics for Tri-Crit-Cont

In this section, we propose some polynomial-time heuristics for Tri-Crit-Cont, which was shown NP-hard (see Theorem 1). We start by outlining the general principles that have guided the design before exposing the details for each heuristic.

6.1 General principles

The heuristics work in two steps: first we apply a simple list scheduling algorithm in order to map the DAG onto the p processors. More precisely, we apply a critical-path list scheduling, which assigns 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. At the end of this first step, each task has been mapped on a processor, and it is scheduled for a single execution at maximum speed.

Then, the core of the heuristics consists in reducing the energy consumption of the schedule. We do not change the allocation of the tasks during this second step, but we can slow them down and/or re-execute them.

The first energy reduction technique comes from Proposition 1. The idea is to start by searching for the optimal solution of the problem instance without re-execution, a phase that we call *deceleration*: here 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 the 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 tasks to be re-executed, 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. Let r_i be the start time of task T_i in the current configuration, and d_i be its finish time.

Deceleration: We select a set of tasks that we execute at speed $\max(f_{\texttt{rel}}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, 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.

6.2 List of heuristics

In this section, we describe the different energy-reducing heuristics in more details. We first introduce a few notations:

- *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 $f_{\texttt{re-ex}} = \frac{2c}{1+c} f_{\texttt{rel}}$ (note that $f_{\texttt{re-ex}}$ is the optimal speed in the proof of Theorem 1). If it succeeds, it also re-executes at speed $f_{\texttt{re-ex}}$ 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:

$\mathbf{H}f_{\max}$. In this heuristic, tasks are simply executed once and at maximum speed, by the processor assigned to them by the list scheduling heuristic.

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 $\max(f_{\tt rel}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, so that both the reliability and deadline constraints are matched.

A.Greedy. This is a type A heuristic, where we first set the speed of each task to $\max(f_{\tt rel}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, so that both the reliability and deadline constraints are matched (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 $f_{\tt re-ex}$ 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 $\max(f_{\tt rel}, \frac{\max_{i=1..n} d_i}{D} f_{\max})$, just as in the previous heuristic. 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 $f_{\tt re-ex}$ (re-execution); otherwise, we execute T_i only once, at the slowest admissible speed.

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

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

7 Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 6. All the source-code (our heuristics were coded using the programming language OCaml), together with additional results that were omitted due to lack of space, are publicly available at [2].

7.1 Simulation settings

In order to evaluate the heuristics, we have generated DAGs using the random DAG generation library GGEN [15]. Since GGEN does not give 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 number of edges, see [2] for further information. We choose a reliability constant $\lambda_0 = 10^{-5}$ [1]. We obtain identical results when λ_0 varies from 10^{-5} to 10^{-6} (see Figure 4). 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 processor number p and the reliability speed $f_{\mathtt{rel}}$. In fact, the absolute deadline D is irrelevant, and we rather consider the *deadline ratio*:

$$\textsc{DeadlineRatio} = \frac{D}{D_{\mathtt{min}}},$$

where $D_{\mathtt{min}}$ is the execution time of the list scheduling heuristic when executing each task once and at speed $f_{\max}$. 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.

7.2 Simulation results

We first note that when there is only one 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).

7.2.1 Deadline ratio

In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and $f_{\mathtt{rel}} = \frac{2}{3} f_{\max}$. 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.

7.2.2 Number of processors

In this set of simulations, we let $\textsc{DeadlineRatio} \in \{1.2, 1.6, 2, 2.4\}$ and $f_{\mathtt{rel}} = \frac{2}{3} f_{\max}$. 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.

7.2.3 Reliability $f_{\mathtt{rel}}$

In this set of simulations, we let $p \in \{1, 10, 50, 70\}$ and $\textsc{DeadlineRatio} \in \{1, 1.5, 3\}$. In Figure 3, there are four different curves: the line at 1 corresponds to Hno-reex and $\mathrm{H}f_{\max}$, 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 $\textsc{DeadlineRatio} = 1$: there is no liberty of action on the critical paths. However B.SUS-Crit-Slow gives better results due to its ability of slowing more tasks down. When DeadlineRatio is really tight (equal to 1), decreasing the value of $f_{\mathtt{rel}}$ from $f_{\mathtt{rel}} = 1$ down to $f_{\mathtt{rel}} = 0.9$ 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).

7.3 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
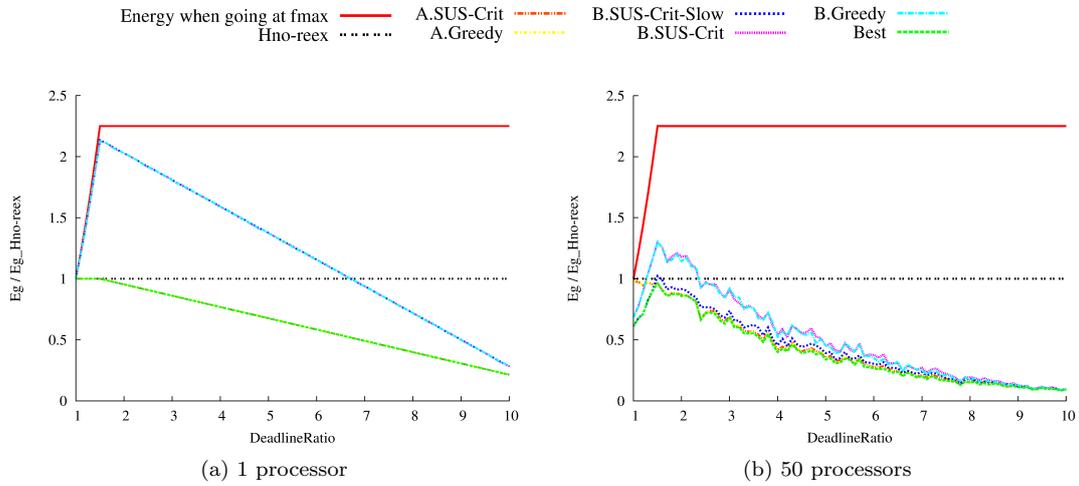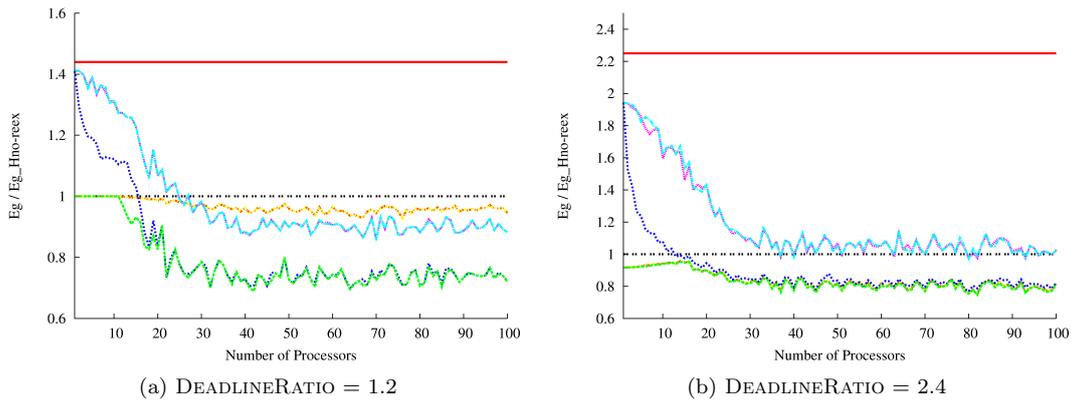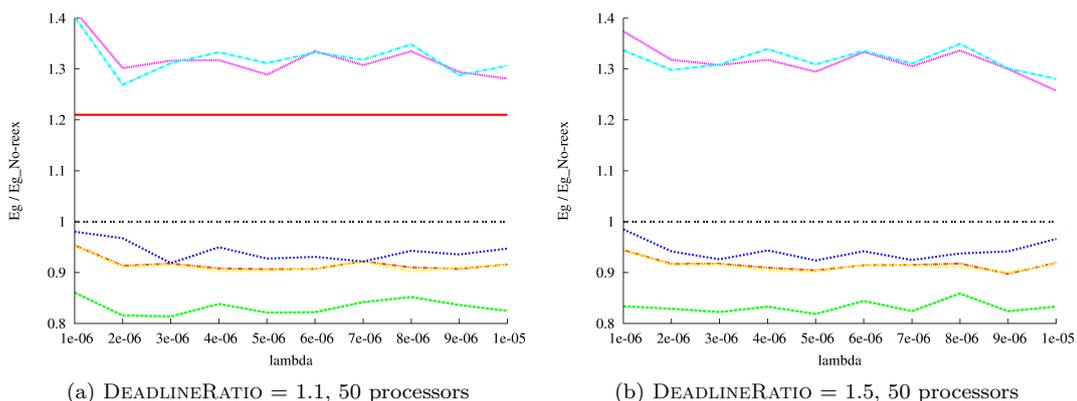
(a) 1 processor (b) 50 processors

Figure 1: Comparative study when the deadline ratio varies.

(a) DEADLINERATIO = 1.2 (b) DEADLINERATIO = 2.4

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

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 clearly better results when there is a small number of processors. When the number of processors increases, there is a difference between small deadline ratio and larger deadline ratio. In particular, when the deadline ratio is small, heuristics of type B will have better results. Here is an explanation: 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.

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.

(a) DEADLINERATIO = 1, 10 processors

(b) DEADLINERATIO = 1, 50 processors
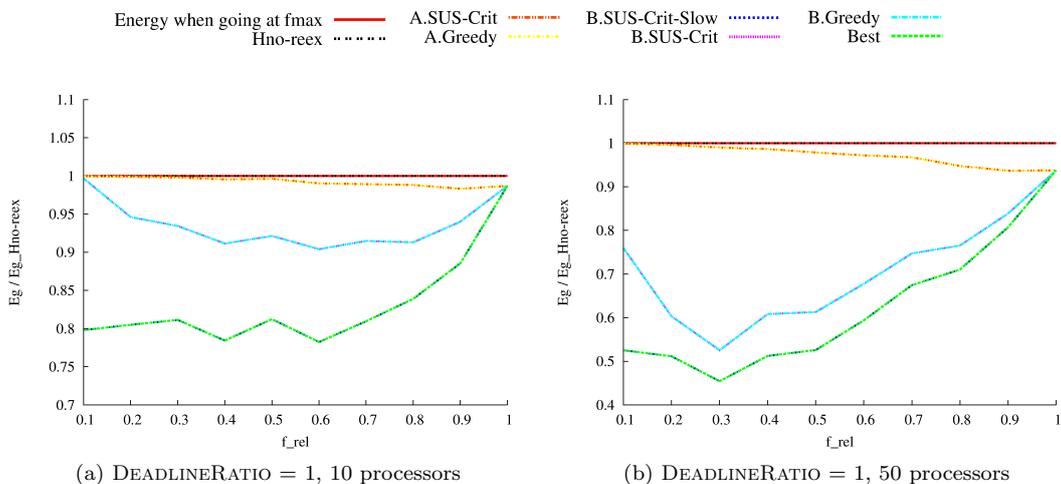
Figure 3: Comparative study when the reliability $f_{\tt rel}$ varies.

(a) DEADLINERATIO = 1.1, 50 processors

(b) DEADLINERATIO = 1.5, 50 processors

Figure 4: Comparative study when the λ_0 varies.

8 Conclusion

In this paper, we have introduced a new energy model for re-execution, that seems more realistic and accurate than the best-case model used in [23]. Coupling this 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? We have stated two variants of this problem, for processor speeds obeying either the CONTINUOUS or the VDD-HOPPING model. We have assessed the intractability of this tri-criteria problem, even when the mapping of tasks to processors is already known. 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 of the list-schedule execution of the application 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.

All the heuristics slow down or re-execute tasks without changing their assignment to processors. In other words, they do not modify the mapping determined by the list-schedule, hence they can also be used when the mapping of the application is already given, e.g. for affinities or security reasons [18, 3].

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. Also, it would be important to assess the impact of the list scheduling heuristic that precedes the energy-reduction heuristic. In other words, the classical critical-path list-scheduling heuristic, which is known to be efficient for deadline minimization, may well be superseded by another heuristic that trades-off execution time, energy and reliability when mapping ready tasks to processors. Such a study could open new avenues for the design of multi-criteria list-scheduling heuristics.

More specifically, we have designed heuristics for the Tri-Crit-Cont model in this paper, but 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 (deadline, 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.

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.

[3] G. Aupy, A. Benoit, F. Dufosse, and Y. Robert. Reclaiming the energy of a schedule: models and algorithms. Research Report 7598, INRIA, France, Apr. 2011. Available at gaupy.org/?paper.

[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] 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.

[8] 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.

[9] 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.

[10] A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *J. Parallel Distrib. Comput.*, 69:326–336, March 2009.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] S. Perarnau. GGEN, a library of Random Graph Generation. http://ligforge.imag.fr/projects/ggen/.

[16] 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.

[17] R. B. Prathipati. Energy efficient scheduling techniques for real-time embedded systems. Master's thesis, Texas A&M University, May 2004.

[18] 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.

[19] W. Rudin. *Principles of mathematical analysis.* McGraw-Hill Book Co., New York, third edition, 1976. International Series in Pure and Applied Mathematics.

[20] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, New York, 1986.

[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] 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.

[23] 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.

[24] D. Zhu, R. Melhem, and D. Mosse. 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.