ERPOT: A quad-criteria scheduling heuristic to optimize the execution time, failure rate, power consumption and temperature in multicores

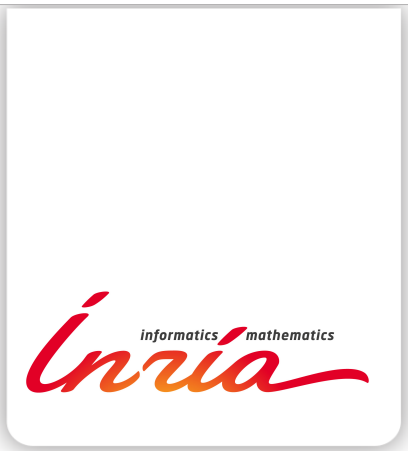
Athena Abdi, Alain Girault, Hamid Zarandi

ERPOT: A quad-criteria scheduling heuristic to optimize the execution time, failure rate, power consumption and temperature in multicores

ABDI Athena, GIRAULT Alain, ZARANDI Hamid

RESEARCH REPORT
N° 9196
July 2018
Project-Teams Spades

ISSN 0249-6399 ISRN INRIA/RR--9196--FR+ENG



ERPOT: A quad-criteria scheduling heuristic to optimize the execution time, failure rate, power consumption and temperature in multicores

Abdi Athena*, Girault Alain†, Zarandi Hamid‡

Project-Teams Spades

Research Report n° 9196 — July 2018 — 35 pages

* Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran.
† Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France.
‡ Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Abstract: We address the problem of computing a static schedule of a DAG of tasks onto an multicore architecture, with the goal of optimizing four criteria: the execution time, the failure rate, the maximum power consumption, and the peak temperature. We propose two methods. The first is a ready list scheduling heuristic called ERPOT (Execution time, failure Rate, POwer consumption and Temperature): it builds a static schedule of the given DAG of tasks onto the given multicore such that its failure rate, power consumption, and temperature remain below three given thresholds, and such that its total execution time is as low as possible. ERPOT replicates actively the tasks to decrease the failure rate, uses Dynamic Voltage and Frequency Scaling to decrease the power consumption, and inserts cooling times to control the peak temperature. The second method uses an Integer Linear Programming (ILP) program to compute an optimal schedule.
We advocate that, when one wants to optimize multiple criteria, it makes more sense to build a set of solutions, each one corresponding to a different tradeoff between those criteria, rather than to build a single solution. This is all the more true when the criteria are antagonistic, which is the case here: for instance, improving the failure rate requires to add some redundancy in the schedule (in our case spatial redundancy), which penalizes the execution time. For this reason, we use ERPOT to build a Pareto front in the 4D space (exec. time, fail. rate, power, temp.), by varying the three thresholds on the failure rate, power, and temperature. Our experimental comparisons show that the schedules produced by ERPOT are on average only 10% worse than the optimal schedules computed by our ILP program, and that ERPOT outperforms the PowerPerf-PET heuristic from the literature by at least 35%.

Key-words: Multicore static scheduling, reliability, failure rate, power consumption, temperature, multi-objective optimization, Pareto front.

ERPOT: Une heuristique d'ordonnancement quadri-critère pour optimiser le temps d'exécution, le taux de défaillance, la puissance électrique et la température sur les multi-cœurs

Résumé : Nous nous attaquons au problème d'ordonnancer statiquement un DAG de tâches sur un processeur multi-cœurs avec comme objectif l'optimisation de quatre critères: le temps d'exécution, le taux de défaillance, la puissance électrique, et la température de crête. Nous proposons deux méthodes. La première est une heuristique de liste appelée ERPOT (Execution time, failure Rate, POwer consumption and Temperature): elle construit un ordonnancement statique du DAG donné sur le multi-cœurs donné de telle sorte que son taux de défaillance, sa puissance électrique et sa température restent au dessous de trois seuils donnés, et que son temps d'exécution soit le plus petit possible. ERPOT utilise la réplication active des tâches pour réduire le taux de défaillance, utilise l'Ajustement Dynamique de la Fréquence et de la Tension (ADFT) pour réduire la puissance électrique, et insère des intervalles d'inactivité pour contrôler la température. La seconde méthode repose sur un Programme Linéaire en Nombres Entiers (PLNE) pour construire un ordonnancement optimal.

Dès que l'utilisateur veut optimiser plusieurs critères, nous préconisons de produire un ensemble de solutions, chacune d'entre elles correspondant à un différent compromis entre ces critères, plutôt que de ne produire qu'une seule solution. Ceci est d'autant plus vrai quand les critères sont antagonistes, ce qui est le cas ici: ainsi, améliorer le taux de défaillance requiert d'ajouter une certaine forme de redondance dans l'ordonnancement (dans notre cas de la redondance spatiale), ce qui pénalise le temps d'exécution. Pour cette raison, nous utilisons ERPOT pour construire un front de Pareto dans l'espace 4D (temps, défaillance, puissance, température), en faisant varier les trois seuils sur le taux de défaillance, la puissance électrique, et la température. Nos évaluations expérimentales montrent que les ordonnancement générés par ERPOT sont en moyenne 10% plus longs que ceux optimaux produits par notre PLNE, et en moyenne 35% plus courts que ceux générés par l'heuristique PowerPerf-PET tirée de la littérature.

Mots-clés : Ordonnancement statique, multi-cœurs, fiabilité, taux de défaillance, température, puissance électrique, optimisation multi-critères, front de Pareto.

Contents

1 Introduction

Multicores are widely used in modern safety critical embedded systems design. Their advantages over super-scalar processor architectures are [1]: lower power consumption, higher performance, and lower design complexity. When designing safety critical applications, many non-functional criteria must be addressed, the most important ones being the *total execution time* (because these systems must react to inputs within a fixed delay), the *failure rate per time unit* (because a failure could have fatal consequences), the *power consumption* (to maximize the autonomy of the system when it operates on battery), and the *temperature* (because of its negative influence on processing speed, failure rate, and power consumption) [1, 2, 3, 4]. Real-life applications that motivate our study are plenty: satellite systems, portable medical devices, full authority digital engine control (FADEC) in aircraft, and so on.

Considering all of these criteria simultaneously during the design phase is very difficult because they have *antagonistic* relationships between themselves [5, 1, 6, 2, 7, 4, 8, 9]. For instance, the total execution time and the failure rate are antagonistic because decreasing the failure rate (or equivalently increasing the reliability) requires some form of redundancy, be it spatial or temporal, which impacts negatively the execution time. Similarly, the execution time and the

temperature are antagonistic because adding idle times to cool the cores obviously has a negative impact on the execution time. Finally, the execution time and the power consumption are antagonistic because reducing the power consumption requires to lower the operating voltage and frequency of the cores, which increases the execution time. Those tradeoffs are easy to grasp (and yet still difficult to address), but other tradeoffs are less obvious: for instance, lowering the operating voltage and frequency of a core (which benefits the power consumption) increases the nominal failure rate per time unit of this core. Indeed, the sensitivity of embedded systems to energy particles leads to an increase of the failure rate at low frequencies). As a consequence, the power consumption and the failure rate are antagonistic too. Failing to take into account these antagonisms would result in bad design choices.

These antagonisms make the design space exploration complex. More importantly, they call for the computation of as many tradeoffs as possible, rather than a single tradeoff. In other words, we advocate that we must produce a *set* of solutions in the 4D space (exec. time, fail. rate, power, temp.) rather than a *single* solution. This raises the issue of how to compare solutions in this 4D space. To do this, we use the notion of *Pareto dominance* (see Section 2). Then, to produce the *Pareto front* in this 4D space, we use a variant of the ε-constraint method [10, 11] coupled with a ready list scheduling heuristic with a specific cost function that accounts for the four criteria.

Although there are several research that take into account some of these parameters, none of them consider these four criteria simultaneously in a joint optimization problem. For instance, some studies ignore completely the reliability [12, 4] or the temperature [2, 9]. Other studies tackle the problem as a hardware/software co-design problem, jointly optimizing the floorplan of the multicore and the schedule of the application task graph to minimize the peak temperature [12], but without considering the failure rate or the reliability.

We propose a static scheduling heuristic method called ERPOT, that optimizes the Execution time, the failure Rate, the POwer consumption, and the Temperature. For a given problem instance, specified as a directed acyclic graph (DAG) of tasks and as an architecture description, ERPOT produces the Pareto front in the 4D space (exec. time, fail. rate, power, temp.). Additionally, we present an ILP program of the defined optimization problem and an exhaustive search program, both of which are used to validate the proposed heuristic and to assess experimentally its approximation ratio. Comparing the results of ERPOT with the optimal results obtained by the ILP program shows that the average difference is less than 10%. But of course, ERPOT is much faster than both the ILP program and the exhaustive search, which both fail to complete even for application graphs of relatively small sizes (8 nodes for the ILP program and 5 nodes for the exhaustive search).

The key contributions of this paper are:

- The ERPOT quad criteria scheduling heuristic, which optimizes the *execution time*, the *failure rate*, the *power consumption*, and the *temperature*. ERPOT uses *active replication*, *dynamic voltage frequency (DVFS)*, and *cooling times insertion* to improve respectively the failure rate, the power consumption, and the temperature.

- A 4D variant of the ε-constraint method [10] to build the Pareto front of the solutions in the 4D space (exec. time, fail. rate, power, temp.). Each point in the Pareto front is obtained by running ERPOT with specific constraints for the failure rate, the power consumption, and the temperature.

- An ILP program of the quad criteria optimization problem to compare the solution computed by ERPOT with the optimal solution.

The rest of this paper is organized as follows. Section 2 recalls the basics about Pareto dominance and how to compute the Pareto front with the ε-constraint method. Section 3 briefly

surveys the related work. Section 4 provides the required preliminaries including the application and architecture models and the interplay between the failure rate, the power consumption, the temperature, and the execution time. Section 5 provides the proposed scheduling heuristic ERPOT, along with its ILP counterpart. Section 6 presents the simulation results. Finally, Section 7 gives some concluding remarks.

2 Pareto optimization

Before presenting the related work, it is necessary to explain the basics about Pareto optimization. Pareto optimization is an efficient solution for solving multi-objective problems. The principle is to explore the entire design space by providing *as many solutions as possible*. Indeed, as soon as one is interested in optimizing more than one criterion, there are *several non-comparable* solutions, e.g., $(42, 13)$ versus $(9, 78)$ in the case of two criteria that must be minimized. Hence it is interesting, from the final user point of view, to have access to all the possible solutions. In particular, this allows to study the *tradeoffs* between these solutions. To compare solutions, we rely on the notion of *dominance* and Pareto optima, presented below in the case of two criteria that must be minimized:

- The point (x, y) *weekly dominates* the point (x', y') iff $(x < x' \wedge y = y') \vee (x = x' \wedge y < y')$. E.g., in Fig. 1(a), x_2 weakly dominates x_1.
- The point (x, y) *strongly dominates* the point (x', y') iff $(x < x' \wedge y < y')$. E.g., in Fig. 1(a), x_3 strongly dominates y_1.
- A point is a *weak Pareto optimum* iff there does not exist another point that strongly dominates it. E.g., in Fig. 1(a), x_1, ..., x_5 are weak Pareto optima.
- A point is a *strong Pareto optimum* iff there does not exist another point that dominates it (weakly or strongly). E.g., in Fig. 1(a), x_2, ..., x_5 are strong Pareto optima.
- The *Pareto front* is the set of all weak and strong Pareto optima.

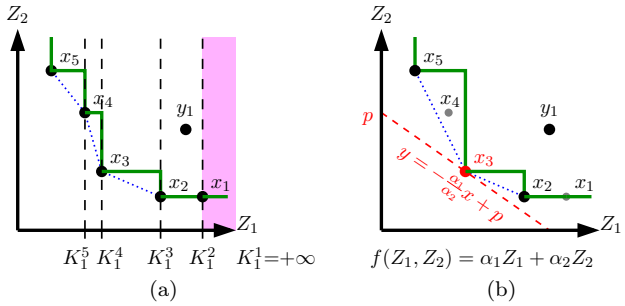


Figure 1: Two transformation methods to compute the Pareto front (2D case): (a) ε-constraint method; (b) aggregation method with a weighted sum.

Building the whole Pareto front and considering all constraints in a multi-criteria problems is a complicated task. To do this, several approaches exist [13], including the *aggregation* method that combines all the criteria in a single cost function, the *hierarchization* method that optimizes one criteria at a time, and the *transformation* method that transforms all the criteria except one into thresholds, and optimizes the remaining criterion under the constraints that the thresholds

are satisfied (this last method is also called "budget optimization"). It is also possible to use population based methods, (e.g., genetic algorithms, particle swarm, ant colony, ...) or the Normal-Boundary Intersection method (NBI) [14].

Varying the cost function in the aggregation method or varying the order of the criteria in the hierarchization method can lead to computing several Pareto points, but not the entire Pareto front. This is a major theoretical drawback. The aggregation method is illustrated in Fig. 1(b) where the aggregation function is the weighted sum $f(Z_1, Z_2) = \alpha_1 Z_1 + \alpha_2 Z_2$. For two given values of the coefficients α_1 and α_2, the Pareto point that is found is the one that minimizes f: geometrically, it is the point that intersects the straight line of slope $-\alpha_1/\alpha_2$ with the smallest value at origin (the p in Fig. 1(b)). Here, the Pareto point that is found is x_3. The problem is that, if the aggregation function is convex as is the case in our example, then the concave portions of the Pareto front will be missed (e.g., the Pareto point x_4). But if the aggregation function is not convex, then there is no guarantee that the computed points are on the Pareto front. For instance, a non-convex aggregation function could return the point y_1. Another problem is the fact that changing the values of the coefficients α_i in the weighted sum might return the same Pareto point as the previous search did, which is inefficient.

Overall, we advocate that the transformation method is an effective method to build the entire Pareto front when used in an iterative way. In the case of two criteria, this process is known as the ε-Constraint Method (εCM) [10] and is illustrated in Fig. 1(a): among the two criteria Z_1 and Z_2, Z_1 is transformed into a constraint. At iteration 1, the threshold for Z_1 is set to $K_1^1 = +\infty$, yielding the Pareto optimum x_1. At iteration 2, the threshold K_1^2 is set to the horizontal coordinate of x_1, therefore excluding the portion of the plane in pink, yielding the Pareto optimum x_2. This process repeats until all the points of the Pareto front have been found (if there is finite number of them), or until some pre-decided number of Pareto point have been found. Under the two conditions that (i) the number of Pareto optima is *finite* and that (ii) the minimization algorithm w.r.t. Z_2 computes the *optimal* result, εCM computes the entire optimal Pareto front.

εCM has been later generalized to more than two criteria in [11], but at a very high computational cost: $k^{m-1}\mathcal{O}(opt)$, where k is the number of points in the Pareto front, m is the number of criteria, and $\mathcal{O}(opt)$ is the complexity of the single criterion optimization algorithm. This computational complexity makes the generalized εCM unfeasible for our problem (if only for the reason that the number of Pareto points is not bounded).

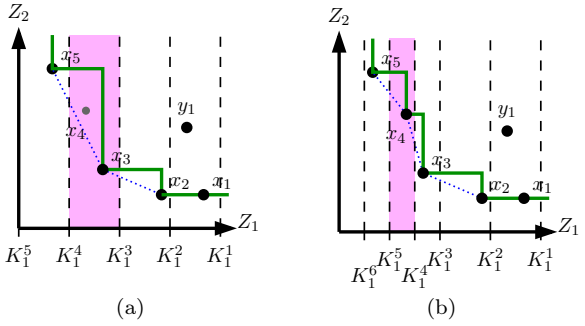


Figure 2: The grid method to compute the Pareto front (2D case): (a) with a coarse regular grid; (b) with an irregular grid.

So we propose instead, for each of the $m-1$ criteria turned into a constraint, to simply divide the useful range of this criterion into p equally spaced intervals, and to invoke a single criterion

optimization algorithm in each of the resulting p^{m-1} zones of the search space. We call this the Grid Method; it is illustrated in Fig. 2(a), where the Z_1 axis is divided into 4 intervals, $[K_1^5, K_1^4)$ to $[K_1^2, K_1^1)$. The resulting complexity therefore becomes $p^{m-1}\mathcal{O}(opt)$. This is still exponential in $m-1$ but the number of intervals p is much less than the number of Pareo points k. The number of intervals can be identical for each of the $m-1$ criteria or not: each range can be divided into p_i intervals (not even necessarily equally spaced), resulting in an overall complexity of $(\prod_{i=1}^{m-1} p_i)\mathcal{O}(opt)$.

The choice of the intervals in each dimension has obviously an impact on the resulting Pareto front. For instance, the grid method illustrated in Fig 2(a) build a Pareto front that does not include the point x_4 because, in the interval $[K_1^4, K_1^3)$ (depicted in pink), the point that minimizes Z_2 is x_3. With a different grid, the point x_4 could be obtained, as shown in Fig. 2(b). On the one hand, using a finer grid will produce a Pareto front with more points, but this can become too costly. On the other hand, using an irregular grid could find more Pareto points but this seems very difficult to control a priori. We therefore advocate that it makes more sense to generate the Pareto front with evenly spaced intervals in each dimension (the number of intervals depending on the time one is ready to spend to compute the Pareto front) and then, in order to improve locally the Pareto front around a particular Pareto optimum, to use either local search methods or to refine the intervals locally around this Pareto optimum. For instance, in Fig. 1, if the Pareto optimum x_3 is identified as an interesting compromise between the criteria Z_1 and Z_2, then the user can either use a local search algorithm around x_3, or he/she can divide the $[K_1^4, K_1^3)$ interval into smaller intervals and invoke again the Z_2 minimization function in these smaller intervals, which will be very likely to find the Pareto optimum x_4.

Finally, note that in Figs 1 and 2, the Pareto front is depicted as a solid green line. It delimits the portion of the plane (above it and on its right) where all the points are dominated by a known Pareto optimum. This differs from the broken line that connects the Pareto optima (depicted in dotted blue), as is demonstrated by Fig. 1(b): the point x_4 is above the dotted blue line, and yet we do not know whether or not is represents a feasible compromise between Z_1 and Z_2, because no Pareto optimum has been found that dominates x_4.

3 Related work

Several related works optimize the executing time, reliability, power consumption, and temperature for applications running on multicores. Most of them consider only two criteria, the execution time and one of the other three criteria. A few related work consider three criteria. However, no existing results consider the four criteria altogether.

Many results address the problem in the context of applications modeled as a set of real-time tasks, usually preemptible, and scheduled by a real-time operating system (RTOS) according to some priority policy (see e.g., [15, 16, 17] to cite only a few). Each task τ_i is defined by a tuple (A_i, C_i, D_i, π_i), where A_i is the arrival time (defined either according to a periodic of sporadic activation model), C_i is the worst-case execution time, D_i is the deadline, and π_i is the priority. Since our application model is totally different, we do not detail these works.

[5] proposes a bi-criteria optimization ready list heuristic algorithm to schedule a DAG of tasks onto an heterogeneous multi-core processor. The algorithm minimizes both the total execution time and the failure rate. Instead of using the system's reliability as an optimization criterion, the authors introduce a new criterion called the Global System Failure Rate (GSFR). The GSFR is computed based on the system's reliability (i.e., the reliability of the schedule on the multicore, computed with classical reliability techniques such as reliability block diagrams) and on the total execution time. The main advantage of the GSFR over the reliability is that it is an

invariant measure of the schedule (see Section 4.3 for details), which makes it suitable to use the transformation method. This allows the authors to use the transformation method to compute the Pareto front in the 2D space (exec. time, GSFR). This method has been generalized in [2] to take into account the power consumption, therefore providing a tri-criteria list scheduling algorithm to optimize the execution time, the GSFR, and the power consumption. The effect of voltage and frequency on the failure rate per time unit of the cores is taken into account. However, it does not take into account the temperature of the cores. ERPOT extends [2] precisely to take into account the temperature.

In [7], the authors propose a bi-criteria genetic algorithm to schedule a DAG of tasks onto a set of identical cores interconnected in a mesh network topology. The algorithm maximizes two criteria, (i) the system reliability (called the "performability") and (ii) the lifetime, under a given energy constraint E_{max} and a given latency constraint P_{max} (the latency is incorrectly denoted "period" by the authors). DVFS is used to lower the total energy consumed. The reliability model, a variant of the Poisson model of Shatz and Wang [18], takes into account the voltage/frequency effect on the failure rate [19]. The lifetime is computed by taking into account failures due to electromigration (EM) thanks to a Weibull distribution. In order to improve the system reliability and the lifetime, some tasks are chosen to be actively replicated; this choice is made by the genetic algorithm. The result is a set of non-dominated solutions in the 2D space (reliability, lifetime). The reliability is improved by increasing the number of replicas, while the lifetime is improved by lowering the temperature. It follows that increasing the number of replicas increases the chip temperature, which in turn decreases the lifetime; in this sense the two criteria are antagonistic. Yet, due to the very high cost of the genetic algorithm, only small DAGs can be scheduled (up to 20 tasks), while we are able to handle DAGS of size greater than 100 tasks. Besides, the effect of the chip temperature on the lifetime and on the reliability is not taken into account. Finally, the leakage power is ignored.

The authors of [4] address the PETOS problem (Performance, Energy, and Temperature Optimized Scheduling), where a DAG of tasks must be scheduled onto a set of M parallel cores operating under K available frequency levels. Because large DAGs are considered, only heuristic algorithms can be used (i.e., neither ILP or exhaustive search algorithms). The authors propose 16 different heuristic, classified according to (i) the core selection strategy and (ii) the frequency selection strategy. First, *none* of the heuristic algorithms proposed in [4] is able to optimize the reliability. Second, the closest heuristic to our work is the PowerPerf-PET heuristic, but it uses *separate* cost functions to select the cores and the frequencies, while we use a unique cost function to select the number of cores, which cores, the frequency level for each core, and the length of each cooling time. Finally, their temperature model is based on measurement rather than an analytic model based on the differential heat propagation equation, and it does not take into account the heat propagation from the neighbor cores. Similarly, their power consumption model is based on measurement, and the effect of the temperature on the power consumption is not taken into account.

Finally, [9] present an energy-efficient fault-tolerant list scheduling heuristic. The application model is a DAG of tasks, and the architecture model is a distributed memory multi-processor with a CAN network. Processors are heterogeneous and equipped with DVFS. Yet, leakage power is ignored. The reliability model is the Poisson model of Shatz and Wang [18], and the effect of DVFS on the failure rate per time unit is taken into account as in [19], but not the temperature. Active replication is applied to each task of the DAG so as to satisfy a given reliability goal for the resulting system (e.g., 0.99). When doing so, the frequencies of the processors the task is mapped to are taken into account. The proposed heuristic minimizes the total energy under this reliability constraint, but the authors do not compute Pareto fronts.

4 Preliminaries

4.1 Application and architecture models

An *application* is modeled as a directed acyclic graph (DAG) $\mathcal{A}lg = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node represents a computing task, and each edge represents data-dependencies among two tasks. All tasks are assumed to be side-effect free. If $X \rightarrow Y$ is a data-dependency, then X is predecessor of Y and Y is successor of X. X is called the source of the data-dependency and Y is called its destination. We also define the sets $pred(X) = \{Y | (Y, X) \in \mathcal{E}\}$ and $succ(X) = \{Y | (X, Y) \in \mathcal{E}\}$. Tasks with no predecessor are called *input* tasks, and those with no successors are called *output* tasks.

Fig. 3(a) shows an example of a DAG with two input tasks (I_1 and I_2), one output task (O_1) and four regular tasks (A, B, C and D).

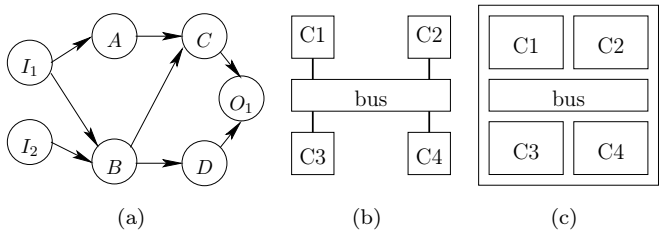


Figure 3: (a) A sample application graph. (b) A sample architecture graph. (c) The corresponding coarse grain floorplan.

An *architecture* is a possibly heterogeneous multicore chip with one or more communication buses. It is modeled as a graph $\mathcal{A}rc = (\mathcal{C}, \mathcal{B}, \mathcal{L})$, where $\mathcal{C}$ is the set of cores, $\mathcal{B}$ is the set of communication buses, and each $e \in \mathcal{L}$ is a pair $(c, b) \in \mathcal{C} \times \mathcal{B}$ specifying that the core c is connected to the bus b. We assume that there exists a path between any two cores c and c'. An example of a target architecture made of four cores and one bus is shown in Fig. 3(b).

We are also given a function $\mathcal{E}xe_{nom}$ that returns the nominal (corresponding to the highest frequency) worst case execution times (WCETs) of all the tasks of $\mathcal{A}lg$ onto all the cores of $\mathcal{A}rc$, as well as the worst case communication times (WCCTs) of all the data-dependencies of $\mathcal{A}lg$ onto all the communication buses of $\mathcal{A}rc$. An intra-core communication takes no time to execute. For the sake of simplicity, all execution times are assumed to be integer numbers.

WCET analysis has been applied with success to real-life single-core processors actually used in embedded systems, with branch prediction [20] or with caches and pipelines [21]. These methods have later been adapted to multicores [22, 23], taking into account the shared resources in the multicore (e.g., the shared memory or the bus).

Finally, our multicores are equipped with a per-core DVFS mechanism. For each core, a set of (voltage,frequency) pairs $\{(V_i, f_i)\}_{1 \leq i \leq \ell}$ is given. For the sake of simplicity, we assume that all the cores have the same set of (voltage,frequency) pairs. The actual execution time of a task τ on a core c depends on the frequency f (in contrast, the buses are assumed to run at a fixed frequency denoted f_b). To ease the computations, we transform the frequencies into scaling factors. E.g., if the set of available frequencies is $\{900\,MHz, 600\,MHz, 300\,MHz\}$, then we use the scaling factors $\{f_{max} = f_3 = 1, f_2 = \frac{2}{3}, f_{min} = f_1 = \frac{1}{3}\}$. As a result, the WCET of task τ at frequency f is given by:

$$\mathcal{E}xe(\tau, c, f) = \lceil \mathcal{E}xe_{nom}(\tau, c)/f \rceil \tag{1}$$

where the $\lceil \cdot \rceil$ function guarantees that $\mathcal{E}xe$ always returns an integer number.

4.2 Static mapping and scheduling

The specifications of the system consists of $\mathcal{A}lg$, $\mathcal{A}rc$, and $\mathcal{E}xe_{nom}$. Implementing such a system involves two steps: First, we must find one or several cores of $\mathcal{A}rc$ to execute each task of $\mathcal{A}lg$, and one or several communication buses of $\mathcal{A}rc$ for each data-dependency: this is the *mapping*. During this phase, we take into account (i) the reliability constraint by choosing how many cores must execute each task, (ii) the power consumption constraint by choosing at what frequency/voltage each component (core or bus) should execute each task and data-dependency, and (iii) the temperature constraint by inserting cooling times whenever necessary. Second, we must compute the starting time for each pair (task,proc) and each pair (data dep.,bus): this is the *scheduling*. This paper solves these two steps *statically*, i.e., at compile time, based on a so-called ready list scheduling heuristic. Finally, as said in the introduction, we schedule under constraints on the failure rate, the power consumption, and the temperature. We note respectively Λ_{obj}, P_{obj}, and T_{obj} these constraints.

4.3 Reliability

Both the cores and the buses are assumed to be *fail-silent*. Classically, we adopt the failure model of Shatz and Wang [18]: failures are *transient*, and the maximal duration of a failure is such that it affects only the current task executing onto the faulty core and not the subsequent tasks (same for the buses); this is the so-called "hot" failure model.

Since the real-time systems we target are safety critical, the occurrence of failures is not acceptable and their reliability must be as close as possible to 1. One of the main causes of system failure are *transient failures* [24], which are commonly modeled by a Poisson distribution with a constant rate denoted λ [25]. Accordingly, the reliability of a single task or data-dependency τ mapped onto a hardware component c (either a core or a bus) running at frequency f is:

$$R(\tau, c, f) = e^{-\lambda_c \cdot \mathcal{E}xe(\tau,c,f)} \tag{2}$$

where λ_c is the *failure rate per time unit* of the hardware component c, and $\mathcal{E}xe(\tau, c, f)$ is the execution time of τ on c at frequency f, computed with Eq. (1). When τ is not replicated, we use Eq.(2). When τ is actively replicated on a set $\mathcal{K}$ of k hardware components numbered $\{c_i\}_{1\leq i\leq k}$, each of them operating at frequency f_{c_i}, its reliability is:

$$R(\tau, \mathcal{K}) = 1 - \left(\prod_{i=1}^{k} \left(1 - e^{-\lambda_{c_i} \cdot \mathcal{E}xe(\tau,c_i,f_{c_i})} \right) \right) \tag{3}$$

Yet, because of the operating frequency f, λ is not constant anymore but is instead a function of the frequency [19]:

$$\lambda_f = \lambda_0 \cdot \rho_f \quad \text{with} \quad \rho_f = 10^{\frac{b(1-f)}{1-f_{min}}} \tag{4}$$

where λ_0 is the nominal failure rate per time unit, ρ_f is the frequency-dependent factor, b is a strictly positive constant that accounts for the susceptibility of hardware to transient faults due to frequency scaling, f is the operational frequency level, and f_{min} is the lowest frequency of the system. Recall that the frequency value f is normalized in the range $[0, 1]$ with $f_{max} = 1$. This is consistent with Eq (1).

Plenty of articles have studied the impact of the temperature on the rate of transient faults [26, 27, 28]. In addition, there are several mechanisms that lead to permanent failures, most notably electro-migration, negative bias temperature instability, stress migration, time-dependent dielectric breakdown, and thermal cycling [29, 3]. All of these phenomena can be characterized by a

failure rate as an exponential function of the temperature. We take into account the effect of the temperature on the failure rate per time unit with the Arrhenius equation [3]:

$$\lambda_T = \lambda_0 \cdot \rho_T \quad \text{with} \quad \rho_T = e^{\frac{-E_a}{K} \cdot (\frac{1}{T(t)} - \frac{1}{T_0})} \tag{5}$$

where again λ_0 is the nominal failure rate per time unit, ρ_T is the temperature-related factor, E_a is the activation energy, K is the Boltzmann's constant, $T(t)$ is the temperature of the system at time t in Kelvin, and T_0 is the initial temperature. Of course, we will also have to take into account the effect of each core's temperature on the other cores (see Section 4.6).

Finally, we combine Eqs (4) and (5) to provide a global equation of the failure rate per time unit as a function of the frequency and the temperature:

$$\lambda_{sys} = \lambda_0 \cdot \rho_f \cdot \rho_T = \lambda_0 \cdot 10^{\frac{b(1-f)}{1-f_{min}}} \cdot e^{\frac{-E_a}{K}(\frac{1}{T(t)} - \frac{1}{T_0})} \tag{6}$$

Since the frequency factor ρ_f and the temperature factor ρ_T are both dimension-less, the dimension of λ_{sys} is the same as λ_0, hence λ_{sys} is also a failure rate per time unit. Fig. 4 illustrates the variations of λ_{sys} as a function of the frequency and the temperature.

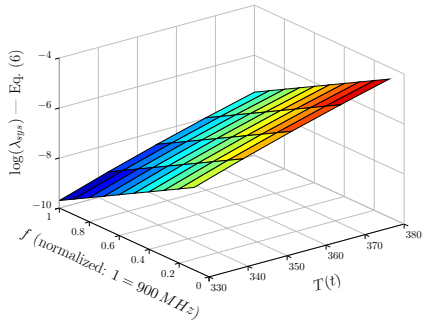


Figure 4: Failure rate per time unit of a core, as a function of the frequency and the temperature ($\lambda_0 = 10^{-5}$, $b = 3$, $E_a = 0.48\,eV$, $T_0 = 298\,K$).

When computing the reliability of a given task or data-dependency τ on a single hardware component c (resp. a set $\mathcal{K}$), we therefore use Eq. (2) (resp. Eq. (3)) by replacing λ by λ_{sys}:

$$R(\tau, c, f, t) = e^{-\lambda_{sys}(c) \cdot \mathcal{E}xe(\tau, c, f)} \tag{7}$$

$$R(\tau, \mathcal{K}, t) = 1 - \left(\prod_{i=1}^{|\mathcal{K}|} \left(1 - e^{-\lambda_{sys}(c_i) \cdot \mathcal{E}xe(\tau, c_i, f_{c_i})} \right) \right) \tag{8}$$

where t is shown to make explicit the dependency of the temperature of each core on the time in λ_{sys}. In the entire paper, we take the temperature at the *task granularity*, i.e., we assume that $T(t)$ remains constant for the entire duration of τ. We will prove in Section 4.4 that doing so is safe w.r.t. the Λ_{obj} constraint.

Eq. (8) is valid only for a single task, not for a full schedule, be it partial or final. For a schedule, we compute the reliability by building the corresponding Reliability Block Diagram [25] (RBD). An RBD is a DAG that starts with a source node S and ends with a destination node D. Between S and D, each of its nodes corresponds to one task (or data-dependency) scheduled on a core (or bus). By definition, an RBD is *operational* iff there exists at least one operational path from S to D. A path is operational iff all the blocks in this path are operational. The

probability that a block is operational is its reliability, computed with Eq. (7). By construction, the probability that an RBD is operational is therefore equal to the reliability of the static schedule it represents.

Computing the reliability in this way assumes that the occurrences of the failures are *statistically independent events*. Concerning hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [30].

When the schedule contains no replication, the RBD is linear. But in general, the schedule does contain replicated tasks, so the structure of its RBD is unspecified, which makes the reliability computation NP-complete [31]. Following [5], the solution we use to prevent this is to insert *routing* tasks (the execution time of which is 0) from the set of replicas of its predecessor task to the set of replicas of its successor task. As a result, the RBD is *serial-parallel*, which makes the reliability computation linear. For any task τ of $\mathcal{A}lg$, all its replicas appear in parallel in the same block of the RBD, and the RBD is composed of all these blocks in sequence (hence the serial-parallel structure).

Consider for instance a simple DAG with two tasks $\sigma \rightarrow \tau$ to be scheduled onto a six-core chip with a single bus b. If σ is replicated twice on cores c_1 and c_2 (its two replicas being denoted σ^1 and σ^2), and τ is replicated twice on cores c_4 and c_5 (its two replicas being denoted τ^1 and τ^2), then the RBD for this schedule will have the form shown in Fig. 5.

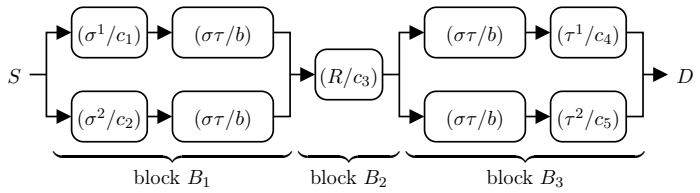


Figure 5: Reliability Block Diagram for a simple schedule with replication.

In practice, our scheduling heuristics will optimize the placement of the routing tasks so as to minimize the total execution time, for instance by mapping R to c_1 or c_2. Finally, the reliability of block B_1 is computed by applying Eq. (8) to task σ followed by the data-dependency $\sigma\tau$:

$$R(B_1, t) = 1 - \left(\prod_{i=1}^{2} \left(1 - e^{-\lambda_{sys}(c_i) \cdot \mathcal{E}xe(\sigma, c_i, f_{c_i}) - \lambda_{sys}(b) \cdot \mathcal{E}xe(\sigma\tau, b, f_b)} \right) \right) \tag{9}$$

The computation of the reliability of block B_3 is similar, and the reliability of block B_2 is equal to 1 since the execution time of the routing task is assumed to be 0. As we can see, thanks to the serial-parallel structure of the RBD, computing the reliability of a schedule (be it partial of final) is compositional, meaning that it can be done block by block.

4.4 Issue related to the Pareto front computation

It has been demonstrated in [5, 2] that using the reliability as a constraint in the ε-constraint method *does not work*. Intuitively, this is because the reliability is not an invariant measure w.r.t. the number of scheduled tasks. Indeed, computing the reliability of a schedule involves, at each mapping decision, a multiplication by a factor that is strictly less than 1: see Eq.(3). This is illustrated in Fig. 6(a), where the horizontal axis counts the task numbers in their mapping order (recall that we use a ready list scheduling algorithm). As long as the reliability is above the threshold R_{obj} (marked with a dashed line), the tasks are not replicated, because this is what minimizes the schedule length; thus the replication level of tasks 1 to 4 is 1 (red line). This

results in a multiplicative factor significantly below 1, which causes the system's reliability to drop (blue line). Once task 4 has been scheduled, the reliability is very close to R_{obj}; this causes the replication level (red line) to skyrocket up to a value sufficient for the multiplying factor to be close enough to 1, so that the system's reliability remains above R_{obj}. We call this the "funnel effect" [2].

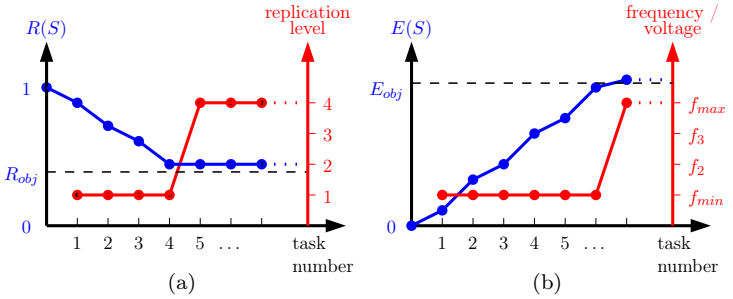


Figure 6: Funnel effect: (a) when using the reliability, (b) when using the energy.

For this reason, instead of the reliability, we use the *Global System Failure Rate* (GSFR) [5]. Intuitively, the GSFR of a possibly partial schedule is the failure rate of the system operating under this schedule *as if* it was a single task mapped on a single core. As a consequence, we schedule under a constraint Λ_{obj} on the GSFR instead of a constraint R_{obj} on the reliability. For a given schedule S, the GSFR is denoted $\Lambda(S)$ and is computed as:

$$\Lambda(S) = \frac{-\log(R(S))}{U(S)} \quad \text{with} \quad U(S) = \sum_{(\tau,c,f)\in S} \mathcal{E}xe(\tau, c, f) \tag{10}$$

where $R(S)$ is the reliability of the schedule S and $U(S)$ is the overall sum of the execution times of the cores in S. The notation $(\tau, c, f) \in S$ means that, in the schedule S, task τ is executed on core c at frequency f. Eq.(10) is equivalent to $R(S) = e^{-\Lambda(S)\cdot U(S)}$, which is the same as Eq. (2) but for a schedule S instead of a single task τ.

One key aspect of Eq. (10) is that it uses $U(S)$ and not the schedule length. There are two reasons behind this choice: first it makes the computation of the GSFR compositional w.r.t. the structure of the schedule (this has been proved in [5]), and second it is consistent with the "hot" failure model.

The consequence of this shift from the reliability to the GSFR is that, from now on, our state space will be the 4D space (exec. time, GSFR, power, temp.). To compute the GSFR of a single task τ mapped on c at frequency f, we first compute its reliability with Eq. (7), and then its GSFR as:

$$\Lambda(\tau, c, f, t) = \frac{-\log(R(\tau, c, f, t))}{\mathcal{E}xe(\tau, c, f)} \tag{11}$$

where t is shown to make explicit the dependency of the temperature of c on the time in $\lambda_{sys}(c)$.

Then, to compute the GSFR of a possibly partial schedule$S$, we build its serial-parallel RBD as explained in Section 4.3, and for each block B_i of S:

- we compute the reliability $R(B_i)$ by applying Eq. (8) to the task and the data-dependencies that belong to B_i;

- we compute the GSFR $\Lambda(B_i)$ by applying Eq. (10):

$$\Lambda(B_i,t) = \frac{-\log(R(B_i,t))}{U(B_i)} \quad \text{with} \quad U(B_i) = \sum_{(\tau,c_j,f_{c_j})\in B_i} \mathcal{E}xe(\tau,c_j,f_{c_j}) \tag{12}$$

  where τ/c_j can represent either a task/core or a data-dependency/bus.

Finally, thanks to the serial-parallel structure of the RBD, we compose the GSFR of all the blocks with the following formula [5]:

$$\Lambda(S) = \frac{\sum_i \Lambda(B_i,t)U(B_i)}{\sum_i U(B_i)} \tag{13}$$

We are now ready to prove that assuming the temperature on each core c_j and on the bus b to remain constant during the duration of each task/data-dependency τ is safe w.r.t. the Λ_{obj} constraint. We prove the result for a generic RBD block B_i made of a task τ replicated k times (hence $|\mathcal{K}| = k$) within a DAG of the form $\sigma{\rightarrow}\tau{\rightarrow}\upsilon$ (see Fig. 7). The reliability of B_i is

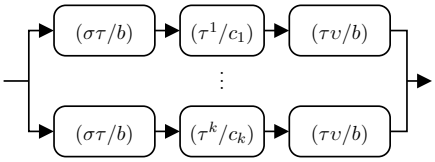


Figure 7: An RBD block B_i made of a task τ replicated k times.

computed by applying Eq. (8) to task τ preceded by the data-dependency $\sigma\tau$ and followed by the data-dependency $\tau\upsilon$:

$$R(B_i,t) = 1 - \left(\prod_{j=1}^{k} \left(1 - e^{-\lambda_{sys}(b)\cdot\mathcal{E}xe(\sigma\tau,b,f_b) - \lambda_{sys}(c_j)\cdot\mathcal{E}xe(\tau,c_j,f_{c_j}) - \lambda_{sys}(b)\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)} \right) \right) \tag{14}$$

Proposition 1 *Let τ be a task mapped onto $\mathcal{K} = \{(c_j, f_{c_j})\}_{1\leq j\leq k}$ such that its block B_i in the RBD has the form shown in Fig. 7). The reliability of B_i is computed with Eq.* (14) *and the GSFR is computed with Eq.* (12)*. For each hardware component c_j (be it a core or a bus) in $\mathcal{K}$, let φ_j/ψ_j be the time at which the operation mapped on c_j (be it a task or a data-dependency) begins/ends. If we take, for each $t \in [\varphi_j, \psi_j]$, $\lambda_{sys}(c_j,t) = \lambda_{sys}(c_j,\psi_j)$, yielding a reliability $R'(B_i)$ and a GSFR $\Lambda'(B_i)$, then $\Lambda'(B_i) \leq \Lambda_{obj}$ implies $\Lambda(B_i,t) \leq \Lambda_{obj}$ (in $\Lambda'(B_i)$ there is no dependence anymore on the time in λ_{sys}).*

Proof: It suffices to prove that $\Lambda(B_i,t) \leq \Lambda'(B_i)$. We have:

$$
\begin{aligned}
& \Lambda(B_i,t) \leq \Lambda'(B_i) \\
\iff\ & -\log(R(B_i,t)) \leq -\log(R'(B_i)) \\
\iff\ & \log(R(B_i,t)) \geq \log(R'(B_i)) \\
\iff\ & R(B_i,t) \geq R'(B_i) \\
\iff\ & \prod_{j=1}^{k} \left(1 - e^{-\lambda_{sys}(b,t)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,t')\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,t'')\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)}\right) \\
& \qquad \leq \prod_{j=1}^{k} \left(1 - e^{-\lambda_{sys}(b,\psi_j)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,\psi'_j)\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,\psi''_j)\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)}\right)
\end{aligned}
$$

For each $1 \leq j \leq k$, there are three time intervals involved, respectively for the data-dependency $\sigma\tau$ on b, for the task τ on c_j, and for the data-dependency $\tau\upsilon$ on b. It follows that there are three upper bounds for these intervals, respectively denoted ψ_j, ψ'_j, and ψ''_j. Let us resume our computation:

$$
\begin{aligned}
& \Lambda(B_i,t) \leq \Lambda'(B_i) \\
\Longleftarrow\ & \forall 1 \leq j \leq k, 1 - e^{-\lambda_{sys}(b,t)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,t')\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,t'')\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)} \\
& \qquad \leq 1 - e^{-\lambda_{sys}(b,\psi_j)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,\psi'_j)\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,\psi''_j)\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)} \\
\iff\ & \forall 1 \leq j \leq k, e^{-\lambda_{sys}(b,t)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,t')\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,t'')\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)} \\
& \qquad \geq e^{-\lambda_{sys}(b,\psi_j)\cdot\mathcal{E}xe(\sigma\tau,b,f_b)-\lambda_{sys}(c_j,\psi'_j)\cdot\mathcal{E}xe(\tau,c_j,f_{c_j})-\lambda_{sys}(b,\psi''_j)\cdot\mathcal{E}xe(\tau\upsilon,b,f_b)} \\
\iff\ & \forall 1 \leq j \leq k, -\lambda_{sys}(b,t)\cdot\mathcal{E}xe(\sigma\tau,b,f_b) - \lambda_{sys}(c_j,t')\cdot\mathcal{E}xe(\tau,c_j,f_{c_j}) - \lambda_{sys}(b,t'')\cdot\mathcal{E}xe(\tau\upsilon,b,f_b) \\
& \qquad \geq -\lambda_{sys}(b,\psi_j)\cdot\mathcal{E}xe(\sigma\tau,b,f_b) - \lambda_{sys}(c_j,\psi'_j)\cdot\mathcal{E}xe(\tau,c_j,f_{c_j}) - \lambda_{sys}(b,\psi''_j)\cdot\mathcal{E}xe(\tau\upsilon,b,f_b) \\
\Longleftarrow\ & \forall 1 \leq j \leq k, \forall (x,c_x,\psi_x,t_x) \in \{(\sigma\tau,b,\psi_j,t),(\tau,c_j,\psi'_j,t'),(\tau\upsilon,b,\psi''_j,t'')\}, \\
& \qquad -\lambda_{sys}(x,t_x)\cdot\mathcal{E}xe(x,c_x,f_{c_x}) \geq -\lambda_{sys}(c_x,\psi_x)\cdot\mathcal{E}xe(\tau_x,c_x,f_{c_x}) \\
\iff\ & \forall 1 \leq j \leq k, \forall (x,c_x,\psi_x,t_x) \in \{(\sigma\tau,b,\psi_j,t),(\tau,c_j,\psi'_j,t'),(\tau\upsilon,b,\psi''_j,t'')\}, \\
& \qquad \lambda_{sys}(x,t_x) \leq \lambda_{sys}(c_x,\psi_x)
\end{aligned}
$$

which is true since λ_{sys} is an increasing function of t_x and $t_x \leq \psi_x$ by definition of $\psi_j, \psi'_j, \psi''_j$. □

This result extends to a full schedule thanks to the serial-parallel structure of the corresponding RBD and to the compositionality of the GSFR computation. Prop. 1 is important because it justifies our assumption that taking the temperature in λ_{sys} at the task granularity is safe w.r.t. the Λ_{obj} constraint.

4.5 Power consumption

The power consumption of a single task (or data-dependency) running on a hardware component is composed of two aspects [19, 32]: (i) the leakage power and (ii) the dynamic power. The former depends on the leakage current, which itself mostly depends on the chip temperature, while the latter depends on the chosen pair (voltage V, frequency f). The overall power consumption P_{sys} is equal to $P_{leak} + P_{dyn}$, computed by Eq. (15):

$$
\begin{cases}
P_{sys}(t) = \alpha \cdot T(t) + \beta_h + \gamma \cdot C_{ef} \cdot V^2 \cdot f & \text{if heating} \\
P_{sys}(t) = \alpha \cdot T(t) + \beta_c + \gamma \cdot C_{ef} \cdot V^2 \cdot f & \text{if cooling}
\end{cases}
\tag{15}
$$

Regarding the leakage power, α, β_h, and β_c are architecture-dependent coefficients and are determined based on the characteristics of the platform; β_h is used in the heating mode and β_c in the cooling mode [8]. Finally, $T(t)$ is the chip temperature at time t, in Kelvin. Regarding the dynamic power, V is the supply voltage, f is the frequency, C_{ef} is the switching capacitance (a constant that depends on the chip technology), and γ is the activity ratio, which varies from 0 (no activity) to 1 (all gates are active at each cycle). In theory, there should be a different γ for each task, and our scheduling algorithm can handle it. In practice, for the sake of simplicity we take an average γ value, identical for all the tasks.

Recall that we take the temperature at the *task granularity*, i.e., we assume that $T(t)$ remains constant for the entire duration of τ. The following property states that doing this is safe w.r.t. the P_{obj} constraint.

Proposition 2 *Let τ be a task or a data-dependency scheduled on a hardware component c at frequency f, starting at time t_0 and finishing at time $t_f = t_0 + \mathcal{E}xe(\tau, c, f)$. The power consumption of c during the execution of τ is computed with Eq. (15). (i) If the temperature increases over the interval $[t_0, t_f]$, then fixing $T(t) = T(t_f)$ is safe w.r.t. the P_{obj} constraint. (ii) If the temperature decreases over the interval $[t_0, t_f]$, then fixing $T(t) = T(t_0)$ is safe w.r.t. the P_{obj} constraint.*

Proof: (i) In the heating mode, the temperature increases during the execution of τ, and when it does, $P_{sys}(t)$ increases too. It follows that assuming $T(t)$ to remain constant over the interval $[t_0, t_f]$ and equal to $T(t_f)$ yields $\forall t, P_{sys}(t) \leq P_{sys}(t_f)$. Therefore, we have:

$$P_{sys}(t_f) \leq P_{obj} \Longrightarrow \forall t \in [t_0, t_f], P_{sys}(t) \leq P_{obj}$$

which proves that assuming that $T(t)$ remains constant and equal to $T(t_f)$ is safe w.r.t. the P_{obj} constraint.

(ii) In the cooling mode, the proof is identical since $T(t)$ decreases so $P_{sys}(t)$ decreases, hence assuming that $T(t)$ remains constant and equal to $T(t_0)$ is safe w.r.t. the P_{obj} constraint. □

From Eq. (15), we can then compute the energy consumed by the system when executing a schedule (possibly partial). However, the same funnel effect as with the reliability occurs if one uses the energy as a constraint in the ε-constraint method [2]. The reason again is that the energy is not an invariant measure w.r.t. the number of scheduled tasks. Indeed, computing the energy consumed by a schedule involves, at each mapping decision, an addition of a term that is strictly positive. This is illustrated in Fig. 6(b): the horizontal axis counts the task numbers in their mapping order; the blue line depicts the cumulative energy consumed by the system while the red line depicts the chosen (frequency,voltage) pair used; up to task 6, the energy is below the energy constraint E_{obj} so everything is fine; however, there is no possibility to schedule task 7 without violating the energy constraint. For this reason, in our multi-criteria scheduling heuristic we use the power consumption, with a constraint P_{obj}, which is an invariant measure w.r.t. the number of scheduled tasks.

4.6 Temperature

The instantaneous temperature of a computing system depends on the power consumption and on the current temperature (and its variations in time). For a given hardware component c (core or bus), it is computed based on the following differential equation [33]:

$$C \cdot \left(\frac{dT_c(t)}{dt} \right) + G\big(T_c(t) - T_{amb}\big) = P(t) \tag{16}$$

where C and G are the architecture-based constants for the heat conductivity, T_c, t, T_{amb}, and P are respectively the temperature of c, the time, the ambient temperature (assumed to be less than T_{obj}[1]), and the instantaneous power consumption of the system. The power consumption is the sum of the dynamic and leakage power, as given by Eq. (15).

For each component c, we wish to take into account the effect of the temperature of its neighbors, according to the chip floorplan. To achieve this, we add a heat transfer term to Eq. (16):

$$C \cdot \left(\frac{dT_c(t)}{dt}\right) + G\big(T_c(t) - T_{amb}\big) + 2D\_heat = P(t) \tag{17}$$

and we use the coarse grain floorplan of Fig. 3(c) (similar to the spatial thermal model and floorplan of [34]) to model this two-dimension heat transfer as:

$$2D\_heat = \sum_{c' \in nbr(c)} \kappa(c, c') \cdot \big(T_c(t) - T_{c'}(t)\big) \tag{18}$$

where $nbr(c)$ is the set of all neighbors of c, T_c is the temperature of c, and $\kappa(c, c')$ is the thermal conductivity between c and c', which depends on their distance and on the chip geometry characteristics (as given in the floorplan).

Combining Eqs. (17) and (18) yields the following differential equation for $T_c(t)$:

$$\begin{aligned} C \cdot \left(\frac{dT_c(t)}{dt}\right) = & -G \cdot \big(T_c(t) - T_{amb}\big) \\ & - \sum_{c' \in nbr(c)} \kappa(c, c') \cdot \big(T_c(t) - T_{c'}(t)\big) \\ & + C_{ef} \cdot V^2 \cdot f + \alpha \cdot T_c(t) + \beta_h \end{aligned} \tag{19}$$

We then proceed as in [8] to re-write Eq. (19) as:

$$\frac{dT_c(t)}{dt} = -A \cdot T_c(t) + B, \qquad \text{with} \tag{20}$$

$$A = \frac{G - \alpha + \sum_{c' \in nbr(c)} \kappa(c, c')}{C}$$

$$B = \frac{G \cdot T_{amb} + \sum_{c' \in nbr(c)} \kappa(c, c') \cdot T_{c'}(t) + C_{ef} \cdot V^2 \cdot f + \beta_h}{C}$$

When computing the evolution of the temperature of c during the execution of τ, we assume that the temperatures of the neighbors remain constant for the entire duration of τ, and equal to their respective temperature at the end of τ. Thanks to the same reasoning as the one made in Section 4.5, this is safe w.r.t. the T_{obj} constraint. It follows that the closed form solution to the differential equation (20) is:

$$T_c(t) = T_\infty^{heat} + \big(T_0 - T_\infty^{heat}\big) \cdot e^{-A(t - t_0)} \tag{21}$$

where $T_\infty^{heat} = B/A$ is the heating steady state temperature and $T_0 = T(t_0)$ is the temperature of the system at t_0. We note $T_c^{heat}(t_0, t)$ the temperature computed with Eq. (21).

[1] If $T_{amb} > T_{obj}$, then putting the component in the idle mode does not allow it to cool down.

When c is idle, the computation of the temperature is identical except that the term $C_{ef} \cdot V^2 \cdot f$ in Eq. (15) disappears and β_h is replaced by β_c, yielding the following closed form:

$$\begin{aligned} B' &= \frac{G \cdot T_{amb} + \sum_{c' \in nbr(c)} T_{c'}(t) \cdot \kappa(c, c') + \beta_c}{C} \\ T_c(t) &= T_\infty^{cool} + \left(T_0 - T_\infty^{cool}\right) \cdot e^{-A(t-t_0)} \end{aligned} \tag{22}$$

where $T_\infty^{cool} = B'/A$ is the cooling steady state temperature and $T_0 = T(t_0)$ is the temperature of the system at t_0, i.e., at the start of the cooling time. We note $T_c^{cool}(t_0, t)$ the temperature computed with Eq. (22).

5 ERPOT: The Proposed Quad-Criteria Optimization Scheduling Heuristic

The optimal mapping of a DAG of tasks on a multicore is a known NP-hard problem [35]. We therefore propose a heuristic algorithm, more precisely a ready list scheduling, for which we formally prove that each computed schedule satisfies the GSFR, power consumption, and temperature constraints (Section 5.2). In addition, in order to assess the performances of our heuristic, we implement an optimal version on top of an ILP solver (Section 5.6).

5.1 General principles of ERPOT

We are given:

(i) a DAG of tasks $\mathcal{A}lg = (\mathcal{V}, \mathcal{E})$,

(ii) a multicore architecture description $\mathcal{A}rc = (\mathcal{C}, \mathcal{B}, \mathcal{L})$ along with the nominal failure rate per time unit λ_0 of each hardware component,

(iii) a function $\mathcal{E}xe_{nom}$ of the nominal WCETs / WCCTs of of all the tasks / data-dependencies of $\mathcal{A}lg$ onto all the cores / buses of $\mathcal{A}rc$,

(iv) a set of frequencies for the cores $\mathcal{F} = \{f_j\}_{1 \leq j \leq \ell}$ and a fixed frequency f_b for the buses, all taken as scaling factors,

(v) three constraints Λ_{obj}, P_{obj}, and T_{obj} respectively on the GSFR, the power consumption, and the temperature,

(vi) and the initial temperature of the chip T_{init}.

The goal is to compute, if it exists, a schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$ such that the three constraints are met and the execution time is minimal (also called C_{max} or schedule length).

In order to keep the GSFR below Λ_{obj}, we use the *active replication* of tasks. As explained in Section 4.4, we compute the reliability of each block with Eq. (8) based on the RBD, and then the GSFR of with Eq. (12).

In order to keep the power consumption below P_{obj}, we use two techniques: (i) on the one hand DVFS, which is available on many modern multicores such as the Intel i7-2600 quad-core or the Samsung Exynos 5422 octa-core; this allows us to lower the P_{dyn} term of Eq. (15); and (ii) on the other hand we try to keep the temperature below T_{obj}, which allows us to lower the P_{leak}

term of Eq. (15). Computing the dynamic power consumption requires to compute the energy consumed by the schedule (be it partial or local), and then to divide by the schedule length. The compositionality issue raised by the GSFR computation also arises here. As demonstrated in [2], this issue can be solved by *over-estimating* the energy consumption each time that the partial schedule has a "gap" at the end, that is, each time one of the cores is idle while the other cores are busy executing their last task. Over-estimation is achieved by computing the energy consumed by such a schedule as if the "gap" was "filled" with a virtual task running at the maximal frequency.

In order to keep the temperature below T_{obj}, we insert *cooling times* to allow the cores to cool down [36, 8, 37] (the buses are always much less loaded than the cores, so they never need to cool down). We follow the same principle as the JUST strategy proposed in [8] for single-core processors, with two differences: in our case the target architecture is a multicore and our objective is to minimize the schedule length under a maximal temperature constraint. The rationale of the JUST strategy is to insert cooling times as late as possible and only when needed, i.e., just in time. Each time we want to schedule a task τ on a core c, we therefore evaluate the temperature of each core in the multicore at the end of this task, taking into account the planned voltage and frequency of τ and the influence of the temperature of the neighbors of c. If it exceeds T_{obj}, then we postpone the starting time of τ by inserting a *cooling time* in order to cool down the core c. The length of the cooling time is the *smallest length* such that the temperature at the end of τ does not exceed T_{obj}.

Recall that a high temperature has a negative effect on the reliability (as shown in Eq. (6)) as well as on the leakage power consumption (see Eq. (15)). This makes it all the more important to limit the maximal temperature.

5.2 Quad-criteria scheduling heuristic algorithm

The proposed heuristic approach is a ready list scheduling algorithm. It works with two lists, the list $\mathcal{R}ready^{(n)}$ of ready tasks and the list $\mathcal{S}ched^{(n)}$ of scheduled tasks, where (n) denotes the current step of the list scheduling. At each step (n), we have $\mathcal{R}ready^{(n)} \cap \mathcal{S}ched^{(n)} = \emptyset$.

In a preliminary phase, we traverse the $\mathcal{A}lg$ graph breadth-first, from the output tasks to the input tasks, in order to compute, for each task τ, the *Longest Execution Path* from τ to the end of the graph, noted $LEP(\tau)$. This notion is similar to the "bottom-level" presented in [38]. Intuitively, $LEP(\tau)$ accounts for all the "future" tasks of τ. For each task τ, it is computed as follows:

- If $succ(\tau) = \emptyset$, then we compute its LEP as $LEP(\tau) = (\sum_{c\in\mathcal{C}} \mathcal{E}xe_{nom}(\tau, c))/|\mathcal{C}|$. The nominal execution time of τ is averaged over all the cores (the set $\mathcal{C}$) since we do not know in advance onto which core τ will be actually scheduled.
- If $succ(\tau) = \{\tau'\}$, then $LEP(\tau) = LEP(\tau') + (\sum_{c\in\mathcal{C}} \mathcal{E}xe_{nom}(\tau, c))/|\mathcal{C}|$. Since τ has only one successor, its nominal execution time is added to the LEP of its only successor (again, averaged over all cores).
- If $succ(\tau) = \{\tau_i\}_{1\leq i\leq k}$ with $k \geq 2$, then $LEP(\tau) = \max_{1\leq i\leq k} LEP(\tau_i) + (\sum_{c\in\mathcal{C}} \mathcal{E}xe_{nom}(\tau, c))/|\mathcal{C}|$. Since τ has more than one successor, its averaged nominal execution time is added to the max of the LEPs of all its successors (again, averaged over all cores).

Still in the preliminary phase, we build the set $2^{\mathcal{C}}$ of all subsets of $\mathcal{C}$, and for each such subset $\{c_i\}_{1\leq i\leq k} \in 2^{\mathcal{C}}$, we build all the possible sets of pairs $\{(c_i, f_j)\}_{1\leq i\leq k, 1\leq j\leq \ell}$, where ℓ is the number of available frequencies. We denote by $\mathcal{Q}$ the set of all such sets of pairs (core,frequency).

In the main phase of ERPOT, we first assign to $\mathcal{R}ready^{(0)}$ the set of input tasks of $\mathcal{V}$, and to $\mathcal{S}ched^{(0)}$ the empty set. Then, at each step (n), we select the *most urgent* task to be scheduled among all the ready tasks, that is, the task τ_{urg} for which $LEP(\tau)$ is the largest: $\tau_{urg} = \text{argmax}_{\tau \in \mathcal{R}ready^{(n)}} LEP(\tau)$.

The next step involves selecting the *best subset of cores* and their associated frequencies to execute τ_{urg}. Each $Q_i \in \mathcal{Q}$ is a potential scheduling choice for τ_{urg}, which we need to evaluate according to our three constraints and our minimization criterion. We denote by Q_{best} the best scheduling choice, by $L^{(n)}$ the schedule length at step (n), thus *before* executing τ on Q_{best}, and by $L^{(n+1)}(\tau, Q_{best})$ the schedule length *after* executing τ on Q_{best}, which we shorten into $L^{(n+1)}$ to avoid heavy notations. Similarly, we denote by $\Lambda^{(n)}$ the GSFR, $E^{(n)}$ the energy, and $T^{(n)}$ the temperature at step (n), again shortened. We further note $\Lambda(\tau, Q_{best})$ the GSFR of the parallel block corresponding to executing τ onto each core of Q_{best}. Recall that we have explained in Section 5.1 that the GSFR of a schedule is computed block by block, thanks to the serial-parallel structure of its RBD. With these notations, Q_{best} is given by the following equation:

$$\begin{aligned} Q_{best} = \underset{Q_i \subseteq \mathcal{Q}}{\text{argmin}} \big\{ L^{(n+1)} \,|\, & \Lambda(\tau, Q_i) \leq \Lambda_{obj} \\ & \wedge \left(E^{(n+1)} - E^{(n)}\right) \leq P_{obj} \cdot \left(L^{(n+1)} - L^{(n)}\right) \\ & \wedge\ T^{(n+1)} \leq T_{obj} \big\} \end{aligned} \tag{23}$$

Eq. (23) might return an empty set Q_{best}. This can occur for three reasons:

1. Either there is no subset of cores Q_i that satisfies the GSFR criterion $\Lambda(\tau, Q_i) \leq \Lambda_{obj}$. In other words, the number of available cores is not sufficient to reach the required GSFR level. The heuristic fails and returns a "no solution" result. Recall that we want to find solutions in the 4D space (exec. time, GSFR, power, temp.). So "no solution" only means that there will be no Pareto point at the coordinates $(\Lambda_{obj}, P_{obj}, T_{obj})$ in the 4D space.

2. Either there is no subset of cores Q_i that satisfies the power consumption criterion $\left(E^{(n+1)} - E^{(n)}\right) \leq P_{obj} \cdot \left(L^{(n+1)} - L^{(n)}\right)$. In other words, the available frequencies are not sufficient to reach the required power consumption level. Like in case 1 above, the heuristic fails and returns a "no solution" result.

3. Or there is no subset of cores Q_i that satisfies the temperature criterion $T^{(n+1)} \leq T_{obj}$. In this case, let $Q'_i = \{c_j \in Q_i \,|\, T^{(n+1)}(c_j) > T_{obj}\}$ and let t_j be earliest time at which τ can start on core c_j. We add to each core $c_j \in Q'_i$ a cooling time of length s_j that starts at t_j, such that s_j is the smallest integer satisfying the inequality:

$$T^{cool}_{c_j}(t_j, s_j) + T^{heat}_{c_j}(t_j + s_j, \mathcal{E}xe(\tau, c_j, f_j)) \leq T_{obj}$$

5.3 Soundness of our scheduling heuristic

We prove in this section four key propositions on the produced schedules, which guarantee that the schedules generated by ERPOT satisfy the Λ_{obj}, P_{obj}, and T_{obj} constraints.

Proposition 3 *Let S be a schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. If each task of $\mathcal{A}lg$ has been scheduled on the subset of cores Q_{best} defined by Eq.* (23)*, thus satisfying the GSFR constraint Λ_{obj}, then the total schedule S will also meet the Λ_{obj} constraint.*

Proof (see [5]): Each task τ_i of $\mathcal{A}lg$ is scheduled onto a subset Q^i_{best} that was selected by Eq. (23). Hence, for all τ_i in $\mathcal{A}lg$, we have $\Lambda(\tau_i, Q^i_{best}) \leq \Lambda_{obj}$. Thanks to the serial-parallel

structure of the RBD corresponding to the schedule S and to the fact that the GSFR is computed compositionally from the RBD, it follows that $\Lambda(S) \leq \Lambda_{obj}$. □

Proposition 4 *Let S be a schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. If each task of $\mathcal{A}lg$ has been scheduled on the subset of cores Q_{best} defined by Eq.* (23)*, thus satisfying the power consumption constraint P_{obj}, then the total schedule S will also meet the P_{obj} constraint.*

Proof (see [2]): The proof follows from Eq. (23) and from the compositionality of the power consumption (as opposed to the energy). Notice that the constraint on the power consumption in Eq. (23) is actually expressed as a constraint between the *energy increase* $(E^{(n+1)} - E^{(n)})$ and the *schedule length increase* $(L^{(n+1)} - L^{(n)})$. The reason is the following: suppose that, at step (n), the most urgent task is τ_i with $Q^i_{best} = \{(c_i, f_i)\}$; suppose also that, in the partial schedule before mapping τ_i, the finish time L_{c_i} on core c_i is such that $L_{c_i} + \mathcal{E}xe(\tau_i, c_i, f_i) \leq L^{(n)}$; in other words, scheduling τ_i on c_i at frequency f_i does *not* increase the current schedule length because there is a "hole" at the end of the schedule of core c_i. Hence $L^{(n)} = L^{(n+1)}$. In contrast, the energy does increase when τ_i is scheduled on c_i at frequency f_i, so $E^{(n+1)} > E^{(n)}$. To overcome this issue, we have proposed in [2] a solution where we "fill" each "gap" at the end of the schedule with a virtual task executing at the maximal frequency f_{max}. It follows the energy consumed by the partial schedule at each step (n) is *over-estimated*.

Thanks to this over-estimation, we prove the desired property by induction on (n). At step (1), the property is verified because the first task is scheduled according to Eq. (23):

$$\left(E^{(1)} - E^{(0)}\right) \leq P_{obj} \cdot \left(L^{(1)} - L^{(0)}\right) \Longleftrightarrow P^{(1)} = \frac{E^{(1)}}{L^{(1)}} \leq P_{obj}$$

Then, our induction hypothesis is:

$$P^{(n)} = \frac{E^{(n)}}{L^{(n)}} \leq P_{obj} \Longleftrightarrow E^{(n)} \leq P_{obj} L^{(n)} \tag{24}$$

Thanks to Eq. (23), we have:

$$\begin{aligned} &\left(E^{(n+1)} - E^{(n)}\right) \leq P_{obj} \cdot \left(L^{(n+1)} - L^{(n)}\right) \\ \Longleftrightarrow &E^{(n+1)} \leq E^{(n)} + P_{obj} L^{(n+1)} - P_{obj} L^{(n)} \end{aligned}$$

Thanks to the induction hypothesis (24), this implies:

$$\begin{aligned} &E^{(n+1)} \leq P_{obj} L^{(n)} + P_{obj} L^{(n+1)} - P_{obj} L^{(n)} \\ \Longleftrightarrow &E^{(n+1)} \leq P_{obj} L^{(n+1)} \\ \Longleftrightarrow &P^{(n+1)} = \frac{E^{(n+1)}}{L^{(n+1)}} \leq P_{obj} \end{aligned}$$

which concludes the proof by induction. □

Proposition 5 *Let S be a schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$ with initial temperature T_{init}. If each task of $\mathcal{A}lg$ has been scheduled on the subset of cores Q_{best} defined by Eq.* (23)*, thus satisfying the temperature constraint T_{obj}, and if $T_{init} \leq T_{obj}$, then the maximum temperature reached during one execution of S starting at T_{init} will also meet the T_{obj} constraint.*

Proof: By hypothesis, $T^{(0)} = T_{init} \leq T_{obj}$. Then, the maximum temperature during S is equal to $\max_{1 \leq i \leq n} T^{(n)}$. Since each scheduling decision satisfies Eq. (23), it follows that $\forall 1 \leq i \leq n$, $T^{(n)} \leq T_{obj}$. As a conclusion we have $\max_{1 \leq i \leq n} T^{(n)} \leq T_{obj}$. □

5.4 Dealing with reactive systems

Propositions 3 and 4 are valid when the schedule is executed once, but also when the schedule is executed repeatedly and infinitely, as is the case for *reactive systems*. What characterizes a reactive system is that it controls some physical device (e.g., a satellite) and that it must continue to do so during the entire life of this physical device. Proposition 5 is valid when the schedule S is executed once, but not when it is repeated infinitely. The reason is due to the difference between the initial temperature T_{init} when the schedule starts and the final temperature T_f when the schedule ends (and also to the fact that the temperature curve depends on the initial temperature, as opposed to the GSFR and the power). Two cases arise:

1. If $T_{init} < T_f \leq T_{obj}$, then executing a second time the same schedule will inevitably *increase* further the temperature, so after some bounded number of executions of this schedule, the multicore temperature will violate the T_{obj} constraint. Recall that the cooling times are static and have been inserted in the schedule based on T_{init}. This is *not safe*.

2. If $T_f < T_{init} \leq T_{obj}$, then executing a second time the same schedule will inevitably *decrease* further the temperature, so after a large number of executions of this schedule, the multicore temperature will drop to the ambient temperature. This is *not optimal*.

Therefore, in order to be safe w.r.t. the T_{obj} constraint and to be optimal, we should guarantee that $T_f = T_{init}$, which can only be achieved by being in Case 1 and then inserting on each core a cooling time until the average temperature of the multicore is equal to T_{init} (because we can cool down the multicore after executing the schedule by inserting a cooling time, while we cannot heat it). Proposition 6 generalizes Proposition 5 to the case of a schedule executed repeatedly.

Proposition 6 *Let S be a schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$ with initial temperature T_{init}, final temperature T_f, and execution time C_{max}. If each task of $\mathcal{A}lg$ has been scheduled on the subset of cores Q_{best} defined by Eq. (23) (thus satisfying the temperature constraint T_{obj}), if $T_{init} \leq T_f \leq T_{obj}$, and if we insert a cooling time of size δ at the end of S such that $T^{cool}(C_{max}, \delta) = T_{init}$, then the maximum temperature reached during an arbitrary number of executions of S starting at T_{init} will also meet the T_{obj} constraint.*

Proof: We prove this property by induction on the number m of executions of S. Let $MaxTemp(k, S)$ denote the maximal temperature during the k-th execution of S.

The case $MaxTemp(1, S) \leq T_{obj}$ is proved by Proposition 5. This first execution of S is followed by a cooling time of size δ, hence $T(C_{max} + \delta) = T_{init}$, which is the start time of the second execution of S.

The induction hypothesis is then:

$$\max_{1 \leq k \leq m} MaxTemp(k, S) \leq T_{obj} \tag{25}$$

The m-th execution of S is followed by a cooling time of size δ, hence $T(m \cdot (C_{max} + \delta)) = T_{init}$, which is the start time of the m+1-th execution of S. Applying the reasoning for $MaxTemp(1, S)$ to the m+1-th execution yields $MaxTemp(m+1, S) \leq T_{obj}$. The proof is then concluded thanks to the induction hypothesis. □

The size δ of the cooling time depends on the difference between T_f and T_{init}. It is obtained

by solving for δ the equation $T^{cool}(C_{max}, \delta) = T_{init}$:

$$\begin{aligned} & \frac{B'}{A} + \left(T_f - \frac{B'}{A}\right) \cdot e^{-A(\delta - C_{max})} = T_{init} \\ \iff \quad & e^{-A(\delta - C_{max})} = \frac{T_{init} - B'/A}{T_f - B'/A} \\ \iff \quad & \delta = C_{max} - \frac{1}{A} \cdot \log\left(\frac{T_{init} - B'/A}{T_f - B'/A}\right) \end{aligned}$$

Since δ must be an integer number, we round it up:

$$\delta = \left\lceil C_{max} - \frac{1}{A} \cdot \log\left(\frac{T_{init} - B'/A}{T_f - B'/A}\right) \right\rceil \tag{26}$$

5.5 Modeling the effect of the adjacent cores on the temperature

In a multicore, multiple cores are located on a single chip at a very short distance from each other, so the temperature of each core impacts the other cores. This is taken into account by Eqs. (18) and (19).

Now, one situation that can arise during our list scheduling algorithm is when the current task $\tau^{(n)}$ is scheduled at step (n) on some core c such that c's neighbors are (partly) idle during the duration of $\tau^{(n)}$. This is illustrated in Fig. 8(a) where task $\tau^{(n)}$ is scheduled on c_2. The risk is that the temperature computed at the end of $\tau^{(n)}$ is under-estimated because the tasks that will be scheduled on the neighbors of c_2 (i.e., c_1 and c_3 in Fig. 8(a)) in a *future* step of the heuristic will not be accounted for. For instance, Fig. 8(b) illustrates the case of a task $\tau^{(n+1)}$ that is scheduled on c_1 at step $(n+1)$, causing an increase of the temperature on c_2 that was not taken into account when we scheduled $\tau^{(n)}$ on c_2.

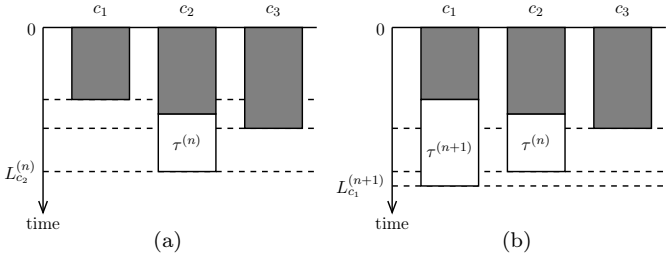


Figure 8: (a) Partial schedule at step (n) and (b) at step $(n+1)$. A white box represents some new task τ such that its vertical length is proportional to $\mathcal{E}xe(\tau, c, f)$. A gray box represents an arbitrary sequence of tasks scheduled during the previous steps.

We solve this issue by adding *virtual tasks* on all the neighbors to *over-estimate* the temperature: each time a task $\tau^{(n)}$ is scheduled on some core c, for each neighbor c' of c such that the current finish time on c' is strictly less than the finish time on c (denoted $L_c^{(n)}$ — note that it can be less than $L^{(n)}$), we add on c' a virtual task that finishes exactly at $L_c^{(n)}$ and that runs at frequency f_{max}. These virtual tasks modify the value of $T_{c'}$ in Eq. (19), therefore guaranteeing that, whatever the future scheduling decisions, the runtime temperature on core c at time $L^{(n)}$ will actually be below the temperature computed during the step (n) of our heuristic. This is illustrated in Fig 9. Of course, when actual tasks are scheduled on these cores c' during future steps, the virtual tasks are removed and the temperature is recomputed accordingly.

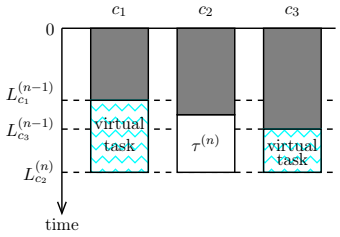


Figure 9: Temperature over-estimation by a adding virtual task to each of the neighbor of c_2 because their respective finish time at step $(n-1)$ was strictly less than $L_{c_2}^{(n)}$.

5.6 Integer Linear Program

We now propose an ILP formulation of our scheduling problem, with the purpose of comparison with the heuristic algorithm presented in Section 5.2. The models and the assumptions used in Section 5.2 are also used here for the ILP program. The decision variables are the following:

$$
\begin{array}{ll}
S_{ik} & \in \mathbb{N} : \text{start time of replica } k \text{ of task } i \\
F_{ik} & \in \mathbb{N} : \text{finish time of replica } k \text{ of task } i \\
Sb_{ik} & \in \mathbb{N} : \text{start time of replica } k \text{ of data dependency } i \\
Fb_{ik} & \in \mathbb{N} : \text{finish time of replica } k \text{ of data dependency } i \\
W & \in \mathbb{N} : \text{total execution time of the application}
\end{array}
$$

$$
x_{ikc} = \begin{cases} 1 & \text{if replica } k \text{ of task } i \text{ is assigned to core } c \\ 0 & \text{otherwise} \end{cases}
$$

$$
x_{ikcfs} = \begin{cases} 1 & \text{if replica } k \text{ of task } i \text{ is assigned to core } c \text{ at} \\ & \text{frequency } f \text{ and after a cooling time} \\ & \text{of } s \text{ time units} \\ 0 & \text{otherwise} \end{cases}
$$

$$
\sigma_{ijkk'} = \begin{cases} 1 & \text{if replica } k \text{ of task } i \text{ starts before replica } k' \\ & \text{of task } j \\ 0 & \text{otherwise} \end{cases}
$$

$$
Y_{iK} = \begin{cases} 1 & \text{if task } i \text{ is replicated } K \text{ times} \\ 0 & \text{otherwise} \end{cases}
$$

$$
B_{ik} = \begin{cases} 1 & \text{if replica } k \text{ of task } i \text{ has an outgoing} \\ & \text{data dependency} \\ 0 & \text{otherwise} \end{cases}
$$

The main objective of our optimization problem is minimizing the total execution time. Then, two kinds of ILP constraints must be formulated. The first kind are the constraints that guarantee the schedulability:

1. Every replica k of task i should be assigned to exactly one core c:

$$\forall i, \forall k, \sum_{c} x_{ikcfs} = 1 \tag{27}$$

2. Every replica k of task i on core c should be assigned to exactly one level of frequency and be preceded by exactly one cooling time (possibly of size 0):

$$\forall i, \forall k, \forall c, \sum_{f,s} x_{ikcfs} = x_{ikc} \tag{28}$$

3. The finish time of every replica k of task i should be less or equal than the total execution time:
$$\forall i, \forall k, F_{ik} \leq W \tag{29}$$

4. The finish time of every replica k of task i is computed based on its execution time and its start time:
$$\forall i, \forall k, F_{ik} = S_{ik} + \sum_{c,f,s} x_{ikcfs} \cdot exe_c(i,c,f,s) + Fb_{ik} \tag{30}$$
where, $exe_c(i,c,f,s)$ is the execution time of task i on core c at the f-th frequency level and after a cooling time of size s: $exe_c(i,c,f,s) = \mathcal{E}xe(i,c,f) + s$.

5. Tasks can not overlap and must obey their precedence order (M is a constant greater than the largest existing number in the ILP program — "big M method" [39]):
$$\forall i \neq j, \forall k, \forall k', \sigma_{ijkk'} + \sigma_{jik'k} \leq 1 \tag{31}$$
$$\forall i, \forall j, \forall k, \forall k', S_{ik} \leq S_{jk'} + (1 - \sigma_{ijkk'}) \cdot M \tag{32}$$
$$\forall i, \forall j, \forall k, \forall k', \forall c, F_{ik} \leq (2 - x_{ikc} - x_{jk'c}) \cdot M + S_{jk'} + (1 - \sigma_{ijkk'}) \cdot M \tag{33}$$
$$\forall i \in pred(j), \forall k, \forall k', F_{ik} \leq S_{jk'} \tag{34}$$
$$\forall i \in pred(j), \forall k, \forall k', \sigma_{ijkk'} = 1 \tag{35}$$

6. If task j is a successor of i and both are assigned to different cores, then this data dependency must be transmitted on the bus:
$$\forall i, \forall k, \forall j \in pred(i), \forall k', \forall c' \neq c, \quad B_{ik} = \bigvee_{c} \left(x_{ikc} \wedge \left(\bigvee_{j,k',c'} x_{jk'c'} \right) \right) \tag{36}$$
where the logical operators $\vee$ and $\wedge$ are linearized [39].

7. The start time of data dependency i is computed based on the first idle time of the bus and on the previous data dependencies transmitted on the bus:
$$\forall i, \forall k, \forall b, Sb_{ik} = \sum_{j,k'} \left((\sigma_{jik'k} \wedge B_{ik} \wedge B_{jk'}) \cdot exe_b(j,b) \right) \tag{37}$$
where $exe_b(j,b)$ is the transmission time of data-dependency j on bus b: $exe_b(j,b) = \mathcal{E}xe(j,b,f_b)$ (recall that buses operate at the fixed frequency f_b, and that we do not insert cooling times on the buses).

8. The finish time of each data dependency is the sum of its start time and its transmission time:
$$\forall i, \forall b, \forall k, Fb_{ik} = Sb_{ik} + B_{ik} \cdot exe_b(i,b) \tag{38}$$

9. Data dependencies must be serialized on the bus:
$$\forall i, \forall k, \forall j \geq i, \forall k', \forall b, \quad Sb_{ik} \leq Sb_{jk'} - exe_b(i,b) + (1 - B_{ik} + \sigma_{ijkk'}) \cdot M \tag{39}$$

The second kind are the ILP constraints that guarantee that the GSFR / power consumption / temperature remain below Λ_{obj}/ P_{obj}/ T_{obj}:

1. The GSFR must be less than or equal to Λ_{obj}:

$$\forall i, \sum_k Y_{ik} = 1 \tag{40}$$

$$\forall i, \forall c, \sum_k x_{ikc} \leq 1 \tag{41}$$

$$\forall i, \sum_{k,c} x_{ikc} = \sum_k k \cdot Y_{iK} \tag{42}$$

$$\forall i, \sum_{k,c,f,s} x_{ikcfs} \cdot GSFR(c,f,s) + \sum_{k,b} B_{ik} \cdot GSFR(b, f_b, 0) \leq \Lambda_{obj} \tag{43}$$

2. The power consumption must be less than P_{obj}:

$$\sum_{i,k,c,f,s} x_{ik\ell mc} \cdot exe_c(i,c,f,s) \cdot P(f,s) + \sum_{i,k,b} B_{ik} \cdot P(f_b, 0) \cdot exe_b(i,b) \leq P_{obj} \cdot W \tag{44}$$

   where $P(f,s)$ is the sum of static and dynamic power consumption when the task runs at frequency f and is preceded by a cooling time of size s.

3. The temperature on each hardware component (cores and bus) must be less than or equal to T_{obj}:

$$\forall i, \forall k, \log(T_\infty^{heat} - T_0) - a \cdot F_{ik} + C \cdot M \geq \log(T_\infty^{heat} - T_{obj}) \tag{45}$$

$$\forall i, \forall k, \log(T_\infty^{cool} - T_0) - a \cdot F_{ik} \leq \log(T_{obj} - T_\infty^{cool}) + (1 - C) \cdot M \tag{46}$$

   where T_0, T_∞^{heat}, and T_∞^{cool} represent respectively the initial temperature at t_0, the heating steady state temperature, and the cooling steady state temperature. Eqs. (45) and (46) are for the cores; for the bus it suffices to replace F_{ik} by Fb_{ik} and to take the value of parameter a corresponding to the bus.

Based on these equations, the main objective of ILP is to minimize the total the execution length (the W variable in our ILP formulation), under the constraints specified by Eqs (27) to (46). In Section 6.4, we will compute the Pareto fronts computed respectively by our quad-criteria heuristic ERPOT and by an ILP solver.

6 Simulation results

We ran several kinds of experiments to evaluate our ERPOT heuristic. In Section 6.1, we assess the influence of the *temperature*, *power consumption*, and *reliability* constraints on the *execution*

time. In Section 6.2, we produce a whole Pareto front for a given problem instance and we visualize it in 3D. In Section 6.3, we compare ERPOT with the PowerPerf-PET scheduling heuristic from [4]. Finally, in Section 6.4, we compare ERPOT with the ILP program of Section 5.6.

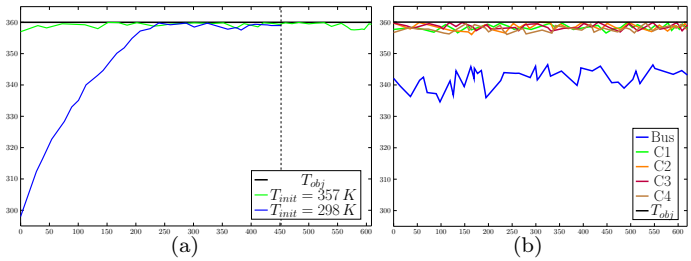
The target multicore chip is shown in Fig. 3(b), with the following parameter values, taken in part from [8] and [7]:

- For each core: $C = 0.03\,JK^{-1}$, $G = 0.3\,WK^{-1}$, $\beta_h = -11\,W$, $\beta_c = -25\,W$, $\alpha = 0.1\,WK^{-1}$.
- For the bus: $C = 0.01\,JK^{-1}$, $G = 0.1\,WK^{-1}$, $\beta_h = -4\,W$, $\beta_c = -8\,W$, $\alpha = 0.04\,WK^{-1}$.
- $C_{ef} = 10^{-8}\,JV^{-2}$ (identical for the cores and the bus).
- Thermal conductivity: $\kappa(bus, c_i) = 0.03\,WK^{-1}$ and $\kappa(c_1, c_2) = \kappa(c_3, c_4) = 0.1\,WK^{-1}$.
- Voltage and frequency pairs for the cores: $\{(900\,MHz, 1.20\,V), (600\,MHz, 1.10\,V), (300\,MHz, 1.06\,V)\}$, corresponding to the scaling factors $\{f_{max} = f_3 = 1, f_2 = \frac{2}{3}, f_{min} = f_1 = \frac{1}{3}\}$.
- Fixed voltage and frequency pair for the bus: $(300\,MHz, 1.06\,V)$, corresponding to the scaling factor $f_b = \frac{1}{3}$.

6.1 Influence of the constraints on the schedules

The experiments in Sections 6.1 and 6.2 are obtained with an $\mathcal{A}lg$ graph consisting of 41 nodes, generated randomly with TGFF [40], and scheduled on the fully connected quad-core chip specified above. The nominal WCETs of the tasks are in the range $[5\,ms, 15\,ms]$; the nominal WCCTs of the data-dependencies are in the range $[3\,ms, 5\,ms]$[2].

Fig. 10(a) shows the variation of the chip temperature in function of the execution time and the effect of the insertion of cooling times in the schedule, for two different values of the initial temperature T_{init}: $298\,K$ and $357\,K$. In both cases, $T_{obj} = 360\,K$, $P_{obj} = 2\,W$, and $\Lambda_{obj} = 10^{-8}$. When $T_{init} = 298\,K$, the temperature increases steadily during a transient phase, and then stabilizes just below T_{obj}, thanks to the cooling times. When $T_{init} = 357\,K$, the temperature remains just below T_{obj} during the whole schedule, again thanks to the cooling times. The initial temperature has a significant impact on the schedule length, from $451\,ms$ for $298\,K$ (indicated by the dashed vertical line) to $608\,ms$ for $357\,K$, a 35% increase.



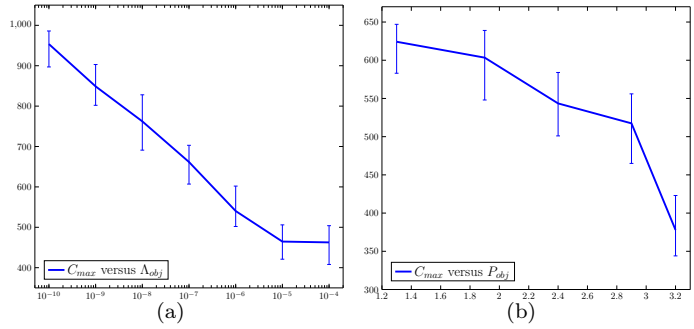
(a) (b)

Figure 10: (a) Evolution of the temperature when $T_{init} = 298\,K$ and $T_{init} = 357\,K$. (b) Evolution of the temperature of each component.

[2]From now on, the time unit will be the millisecond (ms).

Fig. 10(b) depicts the temperature variation of the five hardware components of the chip (bus, C1, C2, C3, and C4) during a schedule produced with the same parameters as Fig. 10(a). The temperatures of the four cores remain in a very small interval, $[356\,K, 360\,K]$, demonstrating the effectiveness of our scheduling heuristic w.r.t. the peak temperature. The bus temperature is significantly below for the simple reason that the bus is often idle. The fact that the temperature variations are very small, both over time and between the cores, is also very good to limit the aging of the chip [7].

Figures 11(a) and 11(b) have been obtained with 50 DAGs generated randomly, each with 50 tasks having an $\mathcal{E}xe_{nom}$ in the range $[3, 12]$, and such that the total sum of the $\mathcal{E}xe_{nom}$ of their tasks is in the range $[540, 560]$. Each point is the average value of the C_{max} over the 50 DAGs, and the vertical bars show the range around the average value.



(a) (b)

Figure 11: (a) Influence of Λ_{obj} and (b) of P_{obj} on the execution time.

As shown in Fig. 11(a), the schedule length increases when Λ_{obj} decreases. This is expected since more replications are required to satisfy the lower failure rate constraint: the two criteria are antagonistic.

As shown in Fig. 11(b), the schedule length increases when P_{obj} decreases. This is expected since lowering the power consumption requires to lower the frequencies used by the cores, which increases the execution time. Again, the two criteria are antagonistic.

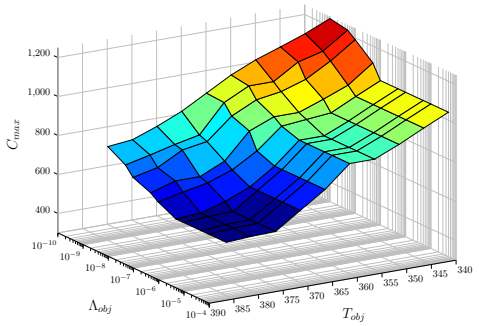
6.2 Pareto fronts obtained with ERPOT

In this Section, we compute the whole Pareto front for an $\mathcal{A}lg$ graph with 41 nodes, onto the fully connected quad-core $\mathcal{A}rc$ graph of Fig. 3(b). Ideally, we would like to visualize this Pareto front in 4D. However, when printed on paper in 3D, the result is very hard to understand. To circumvent this difficulty, we show "snapshots" of the Pareto front, corresponding to various values of P_{obj}. In other words, we display the Pareto front in 3D in the (exec. time, GSFR, temp.) 3D space, and make it vary in the fourth dimension, which is the power consumption. Each snapshot in 3D is therefore a surface that represents the value of C_{max} in function of Λ_{obj} and T_{obj}. We use colors to show the different values of the C_{max}, blue corresponding to lower values and red for higher values.

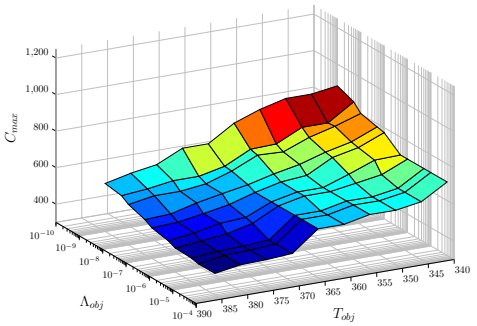
Recall that ERPOT produces Pareto fronts by varying the constraints Λ_{obj}, P_{obj}, and T_{obj} in some range. Here, we have used 10 different values for each criterion:

- $\Lambda_{obj} \in \{5e^{-5}, 1e^{-5}, 5e^{-6}, \ldots, 1e^{-9}\}$;
- $P_{obj} \in \{1.4, 1.7, 2.0, 2.3, 2.6, 2.9, 3.1, 3.4, 3.7, 4.0\}$, all in Watts;

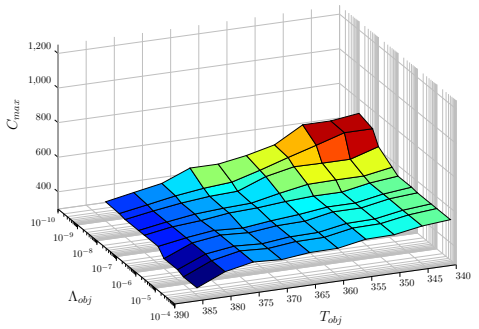
- $T_{obj} \in \{340, 345, \ldots, 385\}$, all in Kelvin.



(a) $P_{obj} = 1.5\,W$



(b) $P_{obj} = 2.5\,W$



(c) $P_{obj} = 3.5\,W$

Figure 12: Pareto fronts in 3D for three different values of P_{obj}: $1.5\,W$, $2.5\,W$, and $3.5\,W$.

Fig. 12 shows the resulting Pareto front in 3D for three different values of P_{obj}: Fig. 12(a) for $P_{obj} = 1.5\,W$, Fig. 12(b) for $P_{obj} = 2.5\,W$, and Fig. 12(c) for $P_{obj} = 3.5\,W$. As Fig. 12(a) shows, a lower value of P_{obj} (here $1.5\,W$) implies high values for the C_{max}, which varies in the range $[572\,ms, 1239\,ms]$. In contrast, for $3.5\,W$, the C_{max} varies in the range $[343\,ms, 718\,ms]$. This figure clearly shows the evolution of the Pareto front when P_{obj} varies. This evolution is expected because the power consumption and the schedule length are antagonistic.

As expected also, when T_{obj} decreases, the C_{max} increases because more cooling times must be inserted and lower frequencies are chosen. For instance, in Fig. 12(a), at $360\,K$ the C_{max} varies in the range $[866\,ms, 1038\,ms]$, while at $340\,K$ it varies in the range $[1041\,ms, 1239\,ms]$.

When Λ_{obj} increases, the C_{max} increase because more tasks must be replicated to compensate for the higher failure rate. Again this is expected because these two criteria are antagonistic. For instance, in Fig. 12(c), at $10\,e^{-5}$ failures per ms, the C_{max} varies in the range $[343\,ms, 533\,ms]$, while it varies in $[411\,ms, 718\,ms]$ at $10\,e^{-9}$.

6.3 Comparison with PowerPerf-PET

We have also compared ERPOT with PowerPerf-PET heuristic from [4] (Algorithm 7), but of course without considering the reliability since PowerPerf-PET addresses only the tri-criteria optimization (exec. time, power, temp.). We have chosen as application graphs the five benchmarks from the E3S suite [41] as well as five DAGs generated randomly with TGFF [40]. For

Benchmark	automotive	consumer	networking	office	telecom	random	random	random	random	random
DAG Size	24	12	13	5	30	40	50	60	70	80
ERPOT (ms)	55	39	48	25	81	232	273	341	378	423
PowerPerf-PET (ms)	91	68	74	39	137	384	438	527	591	667
Improvement (%)	39.56	42.64	35.13	35.89	40.87	39.58	37.67	35.29	36.04	36.58

Table 1: ERPOT against PowerPerf-PET: ERPOT systematically outperforms PowerPerf-PET (by an average of 37.9%).

each DAG, the target architecture was the same quad-core platform as in Section 6.1. For each such problem instance, we have considered the following values for the P_{obj} and T_{obj} constraints:

- $P_{obj} \in \{1.4, 1.7, 2.0, 2.3, 2.6, 2.9, 3.1, 3.4, 3.7, 4.0\}$, all in Watts;
- $T_{obj} \in \{340, 345, \ldots, 385\}$, all in Kelvin.

We have then averaged the C_{max} over the resulting 100 schedules produced by ERPOT and by PowerPerf-PET. Table 1 summarizes the result, where the improvement percentage is computed as:

$$\frac{C_{max}(\text{PowerPerf-PET}) - C_{max}(\text{ERPOT})}{C_{max}(\text{PowerPerf-PET})} \cdot 100$$

As can be seen in Table 1, ERPOT systematically outperforms PowerPerf-PET by at least 35%.

6.4 Evaluation of the ILP model

We have implemented our ILP program (see Section 5.6) in the CPLEX ILOG solver [42] – version 12.6.3, and we have run it on an Intel quad-core i5 CPU with 6GB RAM. The advantage of using an ILP solver is that the result is *optimal*, that is, we obtain the schedule with the *minimal* length under the Λ_{obj}, P_{obj}, and T_{obj} constraints. The inconvenient is that the complexity of finding this optimal schedule is exponential in the size of the problem instance (number of tasks of the $\mathcal{A}lg$ graph plus number of cores times number of frequencies). To be specific, our ILP program was not able to complete its execution for DAGs larger than 9 tasks because the CPLEX solver ran out of memory.

In practice, we have run our ILP program on 10 DAGs randomly generated with TGFF [40], each with 8 tasks, and an homogeneous dual-core with a single bus with three frequency/voltage levels. The WCETs of the tasks are randomly chosen in the range $[3\,ms, 12\,ms]$ while the WCCTs are randomly chosen in the range $[2\,ms, 4\,ms]$. Besides, the cooling times are limited to $1\,ms$. Finally, ten different values of each criterion Λ_{obj}, P_{obj}, and T_{obj} are considered (same as in Section 6.2).

In addition, we have programmed an exhaustive search of the state space, here again in order to find the optimal schedule under the Λ_{obj}, P_{obj}, and T_{obj} constraints. The results obtained by the ILP program and by the exhaustive search are identical.

For each DAG, we have built the full 4D Pareto front with ERPOT and with the ILP program, and then computed the minimum, maximum, and average difference between the two solutions. Table 2 summarizes the results.

On average, the length of the non optimal schedule obtained with our ERPOT heuristic is between 8% and 10% *above* the length of the optimal schedule obtained with the ILP program, which we claim is not too bad. But recall that the ILP solver can only compute the Pareto front for very small DAGs, no larger than 9 tasks.

DAG	1	2	3	4	5	6	7	8	9	10
min (%)	3.2	2.9	3.12	3.08	2.89	3.84	2.71	4.16	3.94	3.3
max (%)	23.74	24.1	25.6	22.38	24.67	25.1	25.47	23.74	22.69	26.38
avg. (%)	8.38	9.26	8.17	8.55	8.49	9.43	9.07	9.94	8.92	9.33

Table 2: ERPOT against ILP: ILP systematically outperforms ERPOT (by an average of 8.95%).

7 Conclusion

We have presented a novel quad-criteria distributed scheduling heuristic called ERPOT (for Execution time, Reliability, POwer consumption and Temperature), which minimizes the schedule length under three constraints: the power consumption, the maximal temperature, and the Global System Failure Rate (GSFR, which generalizes the classical failure rate per time unit of hardware elements to a whole schedule on a multicore architecture). These four criteria are all crucial to optimize for embedded systems. By varying the three constraints and invoking repeatedly our ERPOT heuristic, we are able to compute the whole Pareto front in the 4D space (exec. time, fail. rate, power, temp.).

The failure rate constraint is met by adding active replica in the schedule: the smaller the failure rate, the higher the number of replicas. Hence the failure rate and the schedule length are antagonistic criteria. The power consumption constraint is met by using Dynamic Voltage and Frequency Scaling (DVFS): the smaller the power consumption, the lower the voltage and the frequency. Hence the schedule length and the power consumption are antagonistic criteria. Finally, the temperature constraint is met by inserting cooling times in the schedule (but also by lowering the voltage): the lower the temperature, the longer and more frequent the cooling times. Hence the schedule length and the temperature are antagonistic criteria.

The antagonisms between the criteria already make the scheduling problem quite complex. But there are other interplays that must also be taken into account. For instance, lowering the voltage makes the hardware sensitive to lower energy particles, thereby increasing the nominal failure rates of the hardware components of the target architecture. ERPOT is the first scheduling heuristic able to take into account all those antagonisms.

Extensive experimental results show that our scheduling heuristic works very well: (i) on small application graphs, ERPOT is outperformed on averaged by less than 10% by an ILP program that produces the *optimal* Pareto fronts; (ii) on large application graphs, both synthetic and real-life, ERPOT outperforms the PowerPerf-PET scheduling heuristic by at least 35%.

The largest deviations between ERPOT and ILP occur when the Λ_{obj}, P_{obj}, and T_{obj} constraints are the more stringent. The reason is that the ILP program makes better choices between inserting cooling times and lowering the frequency/voltage. This hints at potential avenues for future improvements of ERPOT.

References

[1] L. Torres, P. Benoit, G. Sassatelli, M. Robert, F. Clermidy, and D. Puschini, "An introduction to multi-core system on chip – trends and challenges," in *Multiprocessor System-on-Chip*, pp. 1–21, Springer Science Business Media, Nov. 2010.

[2] I. Assayad, A. Girault, and H. Kalla, "Tradeoff exploration between reliability, power consumption, and execution time for embedded systems," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 3, pp. 229–245, 2013.

[3] J. E. D. E. Council, "Failure mechanisms and models for semiconductor devices," Tech. Report JEP 122-H, JEDEC, Aug. 2016.

[4] H. Sheikh and I. Ahmad, "Sixteen heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-cores," *ACM Trans. on Parallel Computing*, vol. 3, pp. 1–29, Aug. 2016.

[5] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. on Dependable and Secure Computing*, vol. 6, pp. 241–254, Oct. 2009.

[6] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, "Thermal performance challenges from silicon to systems," *Intel Technology Journal*, vol. Q3, 2000.

[7] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," in *Design, Automation & Test in Europe, DATE'14*, (Dresden, Germany), Mar. 2014.

[8] P. Kumar and L. Thiele, "Thermally optimal stop-go scheduling of task graphs with real-time constraints," in *Asia and South Pacific Design Automation Conference, ASP-DAC'11*, IEEE, Jan. 2011.

[9] G. Xie, Y. Chen, and X. Xiao, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. on Sustainable Computing*, June 2017.

[10] Y. Haimes, L. Lasdon, and D. Wismer, "On a bicriterion formulation of the problems of integrated system identification and system optimization," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 1, pp. 296–297, 1971.

[11] M. Laumanns, L. Thiele, and E. Zitzler, "An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method," *European J. of Operational Research*, vol. 169, no. 3, pp. 932–942, 2006.

[12] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design," *The Journal of VLSI Signal Processing*, vol. 45, pp. 177–189, Dec. 2006.

[13] V. T'Kindt and J. Billaut, *Multicriteria Scheduling – Theory, Models and Algorithms.* Springer, 2006.

[14] I. Das and J. Dennis, "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems," *SIAM J. Opt.*, vol. 8, pp. 631–657, Mar. 1998.

[15] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *Design Automation and Test in Europe Conference, DATE'09*, (Nice, France), pp. 51–56, Mar. 2009.

[16] X. Qin, W. Wang, and P. Mishra, "TCEC: Temperature and energy-constrained scheduling in real-time multitasking systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, pp. 1159–1168, Aug. 2012.

[17] Y. Ma, T. Chantem, and R. P. Dick, "Improving system-level lifetime reliability of multicore soft real-time systems," *IEEE Trans. Very Large Scale Integration Systems*, vol. 25, pp. 1895–1905, Mar. 2017.

[18] S. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Reliability*, vol. 38, pp. 16–26, Apr. 1989.

[19] D. Zhu, R. G. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *ICCAD'04*, pp. 35–40, IEEE / ACM, 2004.

[20] A. Colin and I. Puaut, "Worst case execution time analysis for a processor with branch prediction," *Real-Time Systems*, vol. 18, no. 2/3, pp. 249–274, 2000.

[21] H. Theiling, C. Ferdinand, and R. Wilhelm, "Fast and precise WCET prediction by separate cache and path analyses," *Real-Time Systems*, vol. 18, pp. 157–179, May 2000.

[22] S. Altmeyer, R. Davis, L. Indrusiak, C. Maiza, V. Nélis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *International Conference on Real Time Networks and Systems, RTNS'15*, (Lille, France), pp. 129–138, ACM, Nov. 2015.

[23] H. Rihani, M. Moy, C. Maiza, R. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *International Conference on Real-Time Networks and Systems, RTNS'16*, (Brest, France), pp. 67–76, ACM, Oct. 2016.

[24] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, pp. 11–33, Jan. 2004.

[25] H. Balaban, "Some effects of redundancy on system reliability," in *National Symposium on Reliability and Quality Control*, (Washington (DC), USA), pp. 385–402, Jan. 1960.

[26] D. Rossi, M. Omana, C. Metra, and A. Paccagnella, "Impact of bias temperature instability on soft error susceptibility," *IEEE Trans. Very Large Scale Integration Systems*, vol. 23, pp. 743–751, apr 2015.

[27] A. M. Fard, M. Ghasemi, and M. Kargahi, "Response-time minimization in soft real-time systems with temperature-affected reliability constraint," in *2015 CSI Symposium on Real-Time and Embedded Systems and Technologies, RTEST'15*, IEEE, oct 2015.

[28] S. Hsueh, R. Huang, and C. Wen, "TASSER: A temperature-aware statistical soft-error-rate analysis framework for combinational circuits," in *Fifteenth International Symposium on Quality Electronic Design*, IEEE, mar 2014.

[29] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Exploiting structural duplication for lifetime reliability enhancement," in *ISCA*, pp. 520–531, IEEE, 2005.

[30] J. Knight and N. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming," *IEEE Trans. Software Engin.*, vol. 12, no. 1, pp. 96–109, 1986.

[31] P. Jensen and M. Bellmore, "An algorithm to determine the reliability of a complex system," *IEEE Trans. Reliability*, vol. 18, pp. 169–174, Nov. 1969.

[32] M. Moy, C. Helmstetter, T. Bouhadiba, and F. Maraninchi, "Modeling power consumption and temperature in TLM models," *Leibniz T. on Embedded Systems*, vol. 3, no. 1, pp. 1–29, 2016.

[33] F. Kreith, *CRC Handbook of Thermal Engineering*. Mechanical and Aerospace Engineering Series, CRC Press, 1999.

[34] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *IEEE Trans. Very Large Scale Integration Systems*, vol. 19, no. 10, pp. 1884–1897, 2011.

[35] M. Garey and D. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman Company, 1979.

[36] A. S. Hartman, D. E. Thomas, and B. H. Meyer, "A case for lifetime-aware task mapping in embedded chip multiprocessors," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES/ISSS'10*, ACM, 2010.

[37] Z. Wang, S. Ranka, and P. Mishra, "Efficient task partitioning and scheduling for thermal management in multicore processors," in *ISQED'15*, IEEE, 2015.

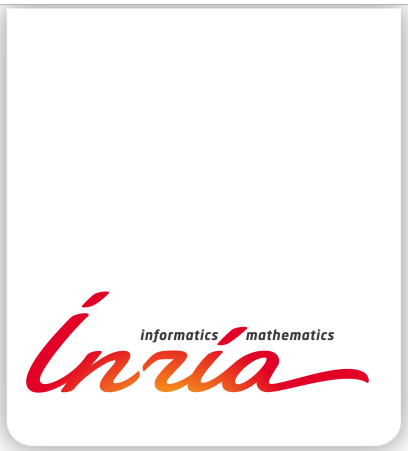
[38] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[39] B. McCarl and T. Spreen, *Applied Mathematical Programming Using Algebraic Systems*. College Station (TX), USA: Texas A&M University, 2007.

[40] "Task graphs for free." `http://ziyang.eecs.umich.edu/~dickrp/tgff`. Accessed: 2016-10-22.

[41] "Embedded microprocessor benchmark consortium." `http://www.eembc.org`.

[42] IBM, "ILOG CPLEX optimizer." `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`. Accessed: 2016-10-22.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399