Tradeoff exploration between reliability, power consumption, and execution time for embedded systems

Ismail Assayad, Alain Girault, Hamoudi Kalla

Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Tradeoff exploration between reliability, power consumption, and execution time for embedded systems

The TSH tricriteria scheduling heuristic

Ismail Assayad · Alain Girault · Hamoudi Kalla

Abstract For autonomous critical real-time embedded systems (e.g., satellite), guaranteeing a very high level of reliability is as important as keeping the power consumption as low as possible. We propose an off-line scheduling heuristic which, from a given software application graph and a given multiprocessor architecture (homogeneous and fully connected), produces a static multiprocessor schedule that optimizes three criteria: its *length* (crucial for real-time systems), its *reliability* (crucial for dependable systems), and its *power consumption* (crucial for autonomous systems). Our tricriteria scheduling heuristic, called TSH, uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling* to lower the power consumption. We demonstrate the soundness of TSH. We also provide extensive simulation results to show how TSH behaves in practice: Firstly, we run TSH on a single instance to provide the whole Pareto front in 3D; Secondly, we compare TSH versus the ECS heuristic (Energy-Conscious Scheduling) from the literature; And thirdly, we compare TSH versus an optimal Mixed Linear Integer Program.

Keywords Embedded systems, multicriteria optimization, reliability, power consumption, DVFS, multiprocessor scheduling, Pareto front.

1 Introduction

1.1 Motivations

Autonomous critical real-time embedded applications are commonly found in embedded devices such as satellite systems. Because they are real-time systems, their execution time must be as low as possible to guarantee that the system interacts with its environment in a timely way. Because they are critical, their reliability must be as close as 1 as possible, typically above $1 - 10^{-9}$. And because they are autonomous, their

A short version of this article has been published in the proceedings of the SAFECOMP'11 International Conference, September 2011, Napoli, Italy.

Ismail Assayad
ENSEM (RTSE team), University Hassan II, Casablanca, Morocco.

Alain Girault
INRIA and Grenoble University (POP ART team and LIG lab), Grenoble, France.

Hamoudi Kalla
University of Batna (LaSTIC lab, REDS team), Batna, Algeria.

power consumption must be as low as possible. The main problem when addressing these issues is that they are *antagonistic*. Intuitively, lowering the probability of failure requires some form of redundancy, meaning more computing load. This is antagonistic to achieving the lowest possible execution time. In the same manner, lowering the power consumption is usually achieved by lowering the voltage and frequency operating point of the processors, which means that the same software function will take more time to execute. Finally, lowering the voltage and frequency operating point also has an impact of the failure rate of processors, because lower voltage leads to smaller critical energy, hence the system becomes sensitive to lower energy particles. As a result, the failure probability increases. These three antogonisms make the problem very challenging.

In order to offer the best compromises between these three measures, we present an off-line scheduling heuristic that, from a given software application graph and a given multiprocessor architecture, produces a static multiprocessor schedule that optimizes three criteria: its *schedule length* (crucial for real-time systems), its *reliability* (crucial for dependable systems), and its *power consumption* (crucial for autonomous systems). We target homogeneous distributed architecture, such as multicore processors. Our tricriteria scheduling heuristic uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling (DVFS)* to lower the power consumption.

1.2 Multicriteria optimization

Let us address the issues raised by multicriteria optimization. Figure 1 illustrates the particular case of two criteria, Z_1 and Z_2, that must be minimized. For the clarity of the presentation, we stick here to two criteria but this discussion extends naturally to any number of criteria. In Figure 1, each point x^1 to x^7 represents a solution, that is, a different tradeoff between the two criteria. The points x^1, x^2, x^3, x^4, and x^5 are called *Pareto optima* [28]. Among those solutions, the points x^2, x^3, and x^4 are called *strong Pareto optima* (no other point is *strictly* better on all criteria) while the points x^1 and x^5 are called *weak Pareto optima* (no other point is better on all criteria, possibly not strictly). The set of all Pareto optima is called the *Pareto front*.

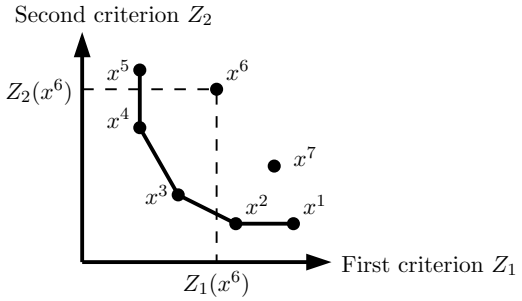


Fig. 1 Pareto front for a bicriteria minimization problem.

It is fundamental to understand that no solution among the points x^2, x^3, and x^4 (the strong Pareto optima) can be said to be the *best* one. Indeed, those three solutions are *non-comparable*, so choosing among them can only be done by the user, depending on the precise requirements of his/her application. But such a user-dependent choice can only be made if we are able to compute the whole Pareto front. If we compute only a single solution, then obviously no choice is possible. This is why we advocate producing, for a given problem instance, the whole Pareto front rather than a single

solution. Since we have three criteria, it will be a *surface* in the 3D space (execution time,reliability,power consumption).

Now, several approaches exist to tackle bicriteria optimization problems (these methods extend naturally to multicriteria) [28]:

1. **Aggregation** of the two criteria into a single one, so as to transform the problem into a classical single criterion optimization one.
2. **Hierarchization** of the criteria, which allows the total ordering of the criteria, and then the solving of the problem by optimizing one criterion at a time.
3. **Interaction** with the user, in order to guide the search for a Pareto optimum.
4. **Transformation** of one criterion into a constraint, which allows the solving of the problem by optimizing the other criterion under the constraint of the first one.

Any multicriteria optimization method that aggregates the criteria (for instance with a linear combination of all the criteria) can only produce one point of the Pareto front, leaving no choice to the user between several tradeoffs. Of course, such a method could be run several times (for instance by changing the coefficients of the linear combination), but there is no way to control what part of the Pareto front will be produced by doing so and the Pareto front is likely to be far from complete.

Similarly, any multicriteria optimization method that hierarchizes the criteria can only produce one point of the Pareto front. For instance, in the case of Figure 1, we could first minimize Z_1 and obtain the subset $\{x^4, x^5\}$ of solutions, and then minimize Z_2 among the solutions in $\{x^4, x^5\}$, thereby obtaining the point x^4. Alternatively, we could first minimize Z_2 and then Z_1, thereby obtaining the point x^2. In both cases, only one point of the Pareto front is obtained.

Finally, we do not want to consider the third class of methods (interaction with the user) because it too would produce a single point of the Pareto front, and also because we want to provide a stand alone multicriteria optimization method.

Contrary to the three first classes of methods, the transformation approach allows one to produce the whole Pareto front, for instance by choosing to take the Z_1 criterion as a constraint, by fixing its maximum value, by minimizing Z_2 under the constraint that Z_1 remains below this value, and by iterating this process with different maximum values of Z_1 so as to produce each time a new point of the Pareto front. This is why the proposed method follows this approach.

1.3 Contributions and outline

The main contribution of this paper is **TSH**, the *first* tricriteria scheduling heuristic able to produce, starting from an application algorithm graph and an architecture graph, a Pareto front in the space (schedule length,reliability,power consumption), and taking into account the impact of voltage on the failure probability. Thanks to the use of active replication, TSH is able to provide any required level of reliability. TSH is an extension of our previous bicriteria (schedule length,reliability) heuristic called BSH [12]. The tricriteria extension presented in this paper is necessary because of the crucial impact of the voltage on the failure probability.

We first present in Section 2 an overview of TSH. Then, in Section 3 we introduce the models that we used, regarding the target architecture, the software application that must be scheduled on it, the execution characteristics of the software elements onto the processing elements, the failure hypothesis, and the power consumption. TSH itself is presented in details in Section 4. In particular, we prove the soundness of TSH by demonstrating that the produced schedules always meet the desired constraint on

the reliability and on the power consumption. Then, in Section 5, we define a mixed integer linear programming model (MILP) for our scheduling problem, in order to compute the optimal Pareto front. Section 6 presents our simulation results, including the comparison with the Energy-Conscious Scheduling heuristic (ECS [18]), and with the optimal Pareto front in the case of small problem instances. Finally, we review the related work in Section 7 and we provide concluding remarks in Section 8.

2 Principle of the method and overview

The approach we have chosen to produce the whole Pareto front involves (i) transforming all the criteria except one into as many constraints, then (ii) minimizing the last remaining criterion under those constraints, and (iii) iterating this process with new values of the constraints.

Figure 2 illustrates the particular case of two criteria Z_1 and Z_2. To obtain the Pareto front, Z_1 is transformed into a constraint, with its first value set to $K_1^1 = +\infty$. The first run involves minimizing Z_2 under the constraint $Z_1 < +\infty$, which produces the Pareto point x^1. For the second run, the constraint is set to the value of x^1, that is $K_1^2 = Z_1(x^1)$: we therefore minimize Z_2 under the constraint $Z_1 < K_1^2$, which produces the Pareto point x^2, and so on. This process converges provided that the number of Pareto optima is bounded. Otherwise it suffices to slice the interval $[0, +\infty)$ into a finite number of contiguous sub-intervals of the form $[K_1^{i+1}, K_1^i]$, resulting in one point for each such interval. That way, the grain of the Pareto front can be improved by reducing the size of the intervals, at the cost of more iterations of the method. Note that each point obtained in this way is not necessarily a point of the Pareto front since it may be dominated by other points.

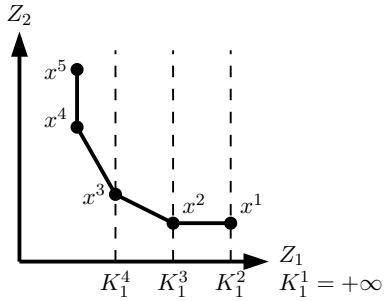


Fig. 2 Transformation method to produce the Pareto front.

Now, the application algorithm graphs we are dealing with are large (tens to hundreds of operations, each operation being a software block), thereby making infeasible exact scheduling methods, or even approximated methods with backtracking, such as branch-and-bound. We therefore chose to use *list scheduling heuristics*, first introduced in [15], and which have demonstrated their good performances for scheduling large graphs [19]. We propose in this paper a tricriteria list scheduling heuristic, called **TSH**, adapted from [12]. TSH improves on [12] by working with three criteria, the schedule length, the reliability, and the power consumption.

Using list scheduling to minimize a criterion Z_2 under the constraint that another criterion Z_1 remains below some threshold K_1^i (as in Figure 2), requires that Z_1 be an *invariant measure*, not a varying one. For instance, the energy is a strictly increasing function of the schedule, in the mathematical sense: if S' is a prefix schedule of S,

then the energy consumed by S is strictly greater than the energy consumed by S'. Hence, the energy *is not* an invariant measure; more precisely it is additive. Figure 3(a) illustrates this fact. The operations are scheduled in the order 1, 2, and so on. Up to the operation number 6, the energy criterion is satisfied: $\forall 1 \leq i \leq 6$, $E(S^{(i)}) \leq E_{obj}$, where $S^{(i)}$ is the partial schedule at iteration (i). But there is no way to prevent $S^{(7)}$ from failing to satisfy the criterion, because whatever the operation scheduled at iteration (7), $E(S^{(7)}) > E_{obj}$. And with list scheduling, it is not possible to backtrack.

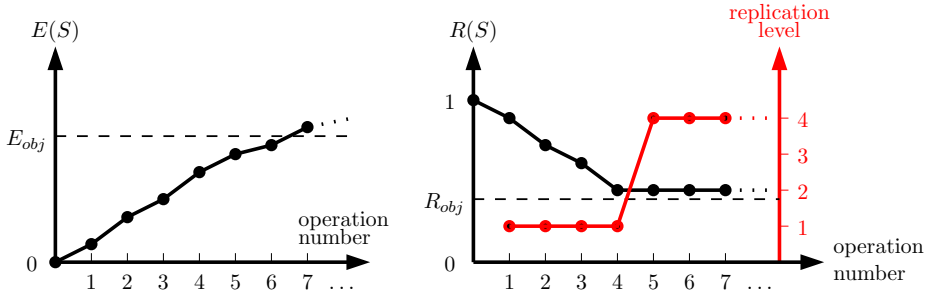


Fig. 3 (a) Why the consumed energy is not an invariant measure; (b) Why the reliability is not an invariant measure and illustration of the funnel effect on the replication level of the operations.

As a consequence, using the energy as a constraint (i.e., $Z_1 = E$) and the schedule length as a criterion to be minimized (i.e., $Z_2 = L$) is bound to fail. Indeed, the fact that all the scheduling decisions made at the stage of any intermediary schedule S' meet the constraint $E(S') < K$ *cannot* guarantee that the final schedule S will meet the constraint $E(S) < K$. In contrast, the power consumption *is* an invariant measure (being the energy divided by the time), and this is why we take the power consumption as a criterion instead of the energy consumption (see Section 3.5).

The reliability too is *not* an invariant measure, because the contribution of each scheduled operation i is a probability in $[0, 1]$, which is multiplied to the reliability of the partial schedule computed so far, $R(S^{(i-1)})$. The consequence is a "so far so good" situation, which results in a "funnel" effect on the replication level of the operations. This is illustrated by Figure 3(b): up to operation 4, the replication level is 1 because this choice minimizes the increase in the schedule length, and the reliability objective is satisfied: $R(S^i) > R_{obj}$ for $i \leq 4$. But at this point, it is not possible to schedule operation 5 with no replication, and at the same time satisfy the reliability objective. However, replicating this operation on all the processors of the target architecture (say 4 for the sake of the example) results in a probability very close to 1, therefore allowing the reliability to decrease only very slightly (Figure 3(b) shows an horizontal line for $R(S)$ after the fifth operation, but actually it decreases very slightly). This is why we take instead, as a criterion, the *global system failure rate per time unit* (GSFR), first defined in [12]. By construction, the GSFR is an invariant measure of the schedule's reliability (see Section 4.1).

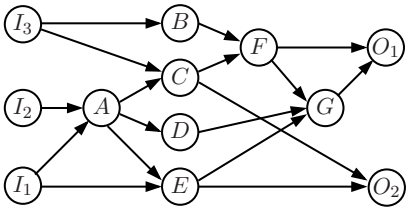
For these reasons, each run of our tricriteria scheduling heuristic TSH minimizes the schedule length under the double constraint that the power consumption and the GSFR remain below some thresholds, noted respectively P_{obj} and Λ_{obj}. By running TSH with decreasing values of P_{obj} and Λ_{obj}, starting with $(+\infty, +\infty)$, we are able

to produce the Pareto front in the 3D space (length,GSFR,power). This Pareto front shows the existing tradeoffs between the three criteria, allowing the user to choose the solution that best meets his/her application needs. Finally, our method for producing a Pareto front could work with any other scheduling heuristic minimizing the schedule length under the constraints of both the reliability and the power.

3 Models

3.1 Application algorithm graph

Embedded real-time systems are *reactive*, and therefore consist of some algorithm executed periodically, triggered by a periodic execution clock. We follow the periodic task model of [17], shown in Figure 4(b). Our model is therefore that of a synchronous application algorithm graph $\mathcal{A}lg$, which is repeated infinitely in order to take into account the reactivity of the modeled system, that is, its reaction to external stimuli produced by its environment. In other words, the body of the periodic loop of Figure 4(b) is captured by the $\mathcal{A}lg$ graph.



(a)

```
Initialize
for each period T do
    Read Inputs
    Compute
    Update Outputs
end for each
```

(b)

Fig. 4 (a) An example of algorithm graph $\mathcal{A}lg$: I_1, I_2, and I_3 are input operations, O_1 and O_2 are output operations, A–G are regular operations; (b) Program model of a periodic real-time task.

$\mathcal{A}lg$ is an *acyclic oriented graph* $(\mathcal{X}, \mathcal{D})$ (See Figure 4(a)). Its nodes (the set $\mathcal{X}$) are software blocks called *operations*. Each arc of $\mathcal{A}lg$ (the set $\mathcal{D}$) is a *data-dependency* between two operations. If $X \triangleright Y$ is a data-dependency, then X is a *predecessor* of Y, while Y is a *successor* of X. The set of predecessors of X is noted $pred(X)$ while its set of successors is noted $succ(X)$. X is also called the *source* of the data-dependency $X \triangleright Y$, and Y is its *destination*.

Operations with no predecessor are called *input* operations (I_1, I_2, and I_3 in Figure 4(a)); they capture the "Read Inputs" phase of the periodic execution loop, each one being a call to a sensor driver. Operations with no successor are called *output* operations (O_1 and O_2); they capture the "Update Outputs" phase, each one being a call to an actuator driver. The other operations (A to G) capture the "Compute" phase and have no side effect.

3.2 Architecture model

We assume that the architecture is an *homogeneous and fully connected multi-processor* one. It is represented by an architecture graph $\mathcal{A}rc$, which is a *non-oriented bipartite graph* $(\mathcal{P}, \mathcal{L}, \mathcal{A})$ whose set of nodes is $\mathcal{P} \cup \mathcal{L}$ and whose set of edges is $\mathcal{A}$ (see Figure 5). $\mathcal{P}$ is the set of processors and $\mathcal{L}$ is the set of communication links. A *processor* is composed of a computing unit, to execute operations, and one or more communication units, to send or receive data to/from communication links. Typically, communication units are DMAs, which present the advantage of sending data in parallel with the

processor. A *point-to-point communication link* is composed of a sequential memory that allows it to transmit data from one processor to another. Each edge of $\mathcal{A}rc$ (the set $\mathcal{A}$) always connects one processor and one communication link. Here we assume that the $\mathcal{A}rc$ graph is *complete*, that is, there exists a communication link between any two processors.

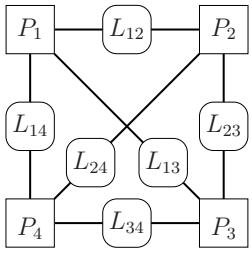


Fig. 5 An example of a distributed memory architecture graph $\mathcal{A}rc$ with four processors, P_1 to P_4, and six communication links, L_{12} to L_{34}.

3.3 Execution characteristics

Along with the algorithm graph $\mathcal{A}lg$ and the architecture graph $\mathcal{A}rc$, we are also given a function $\mathcal{E}xe_{nom} : (\mathcal{X}\times\mathcal{P})\cup(\mathcal{D}\times\mathcal{L}) \to \mathbb{R}^+$ giving the nominal *worst-case execution time* (WCET) of each operation onto each processor and the *worst-case communication time* (WCCT) of each data-dependency onto each communication link. An intra-processor communication takes no time to execute. Since the architecture is homogeneous, the WCET of a given operation is identical on all processors (similarly for the WCCT of a given data-dependency). We call $\mathcal{E}xe_{nom}$ the *nominal* WCET because we will see in Section 3.5 that the actual WCET varies according to the voltage / frequency operating point of the processor.

The WCET analysis is the topic of much work [29]. Knowing the execution characteristics is not a critical assumption since WCET analysis has been applied with success to real-life processors actually used in embedded systems, with branch prediction, caches, and pipelines. In particular, it has been applied to one of the most critical embedded system that exists, the Airbus A380 avionics software [6,27] running on the Motorola MPC755 processor [10,26].

3.4 Static schedules

The graphs $\mathcal{A}lg$ and $\mathcal{A}rc$ are the *specification* of the system. Its *implementation* involves finding a multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. This consists of four functions: the two *spatial allocation functions* Ω_O and Ω_L give respectively, for each operation of and each data-dependency of $\mathcal{A}lg$, the subset of processors and of communication links of $\mathcal{A}rc$ that will execute it; and the two *temporal allocation functions* Θ_O and Θ_L give respectively the starting date of each operation and each data-dependency on its processor or its communication link:

$$\begin{array}{lll} \Omega_O : \mathcal{X} \to 2^{\mathcal{P}} & \text{and} & \Theta_O : \mathcal{X}\times\mathcal{P} \to \mathbb{R}^+ \\ \Omega_L : \mathcal{D} \to 2^{\mathcal{L}} & \text{and} & \Theta_L : \mathcal{D}\times\mathcal{L} \to \mathbb{R}^+ \end{array}$$

In this work we only deal with *static* schedules, for which the functions Θ_O and Θ_L are static, and our schedules are computed *off-line*; i.e., the start time of each operation (resp. each data-dependency) on its processor (resp. its communication link)

is statically known. A static schedule is *without replication* if for each operation X and each data-dependency D, we have $|\Omega_O(X)| = 1$ and $|\Omega_L(D)| = 1$. In contrast, a schedule is *with (active) replication* if for some operation X or data-dependency D, we have $|\Omega_O(X)| \geq 2$ or $|\Omega_L(D)| \geq 2$. The number $|\Omega_O(X)|$ (resp. $|\Omega_L(D)|$) is called the *replication factor* of X (resp. of D). A schedule is *partial* if not all the operations and data-dependencies of $\mathcal{A}lg$ have been scheduled, but all the operations that are scheduled are such that all their predecessors are also scheduled. Finally, the *length* of a schedule is the max of the termination times of the last operation scheduled on each of the processors of $\mathcal{A}rc$ (in the literature, it is also called the makespan). For a schedule S, we note it $L(S)$:

$$L(S) = \max_{P \in \mathcal{P}} \left(\max_{X : P \in \Omega_O(X)} \Theta_O(X, P) + \mathcal{E}xe_{nom}(X, P) \right) \tag{1}$$

In the sequel, we will write $X \in P$ instead of $X : P \in \Omega_O(X)$ for the sake of simplicity. We will also number the processors from 1 to $|\mathcal{P}|$ and use their number in index, for instance p_j (and similarly for the communication links).

3.5 Voltage, frequency, and power consumption

The maximum supply voltage is noted V_{max} and the corresponding highest operating frequency is noted f_{max}. The WCET of any given operation is computed with the processor operating at f_{max} and V_{max} (and similarly for the WCCT of the data-dependencies). Because the circuit delay is almost linearly related to $1/V$ [5], there is a linear relationship between the supply voltage V and the operating frequency f. From now on, we will assume that the operating frequencies are *normalized*, that is, $f_{max} = 1$ and any other frequency f is in the interval $(0, 1)$. Accordingly, we define in Eq (2) a new function $\mathcal{E}xe$ that gives the execution time of the operation or data-dependency X placed onto the hardware component C, be it a processor or a communication link, which is running at frequency f. In other words, f is taken as a scaling factor:

$$\mathcal{E}xe(X, C, f) = \mathcal{E}xe_{nom}(X, C)/f \tag{2}$$

The power consumption P of a single operation or data-dependency placed on a single hardware component is computed according to the classical model found for instance in [21,30]:

$$P = P_s + h(P_{ind} + P_d) \qquad\qquad P_d = C_{ef} V^2 f \tag{3}$$

where P_s is the static power (power to maintain basic circuits and to keep the clock running), h is equal to 1 when the circuit is active and 0 when it is inactive, P_{ind} is the frequency independent active power (the power portion that is independent of the voltage and the frequency; it becomes 0 when the system is put to sleep, but the cost of doing so is very expensive [9]), P_d is the frequency dependent active power (the processor dynamic power and any power that depends on the voltage or the frequency), C_{ef} is the switch capacitance, V is the supply voltage, and f is the operating frequency. C_{ef} is assumed to be constant for all operations; this is a simplifying assumption since one would normally need to take into account the actual switching activity of each operation to compute accurately the consumed energy. However, such an accurate computation is infeasible for the application sizes we consider here.

For processors, this model is widely accepted for average size applications, where C_{ef} can be assumed to be constant for the whole application [30]. For communication

links on a multicore platform, this model is also relevant, as communication links are specialized processing elements [21]. Of course, the coefficients in Eq (3) should be distinct for processors and communication links. We use the following notations:

coefficient	processors	links
frequency independent active power	P^p_{ind}	P^ℓ_{ind}
switch capacitance	C^p_{ef}	C^ℓ_{ef}

Since the architecture is homogeneous, each processor (resp. communication link) has an identical value P^p_{ind} (resp. P^ℓ_{ind}) and similarly an identical value C^p_{ef} (resp. C^ℓ_{ef}). In contrast, since the voltage and frequency varies, each processor p_j has (potentially) a distinct value P^j_d. We do not apply to the communication links so their voltage and frequency remains constant, respectively equal to V_ℓ and f_ℓ.

For a multiprocessor schedule S, we cannot apply directly Eq (3) because each processor is potentially operating at a different V and f, which vary over time. Instead, we must compute the total energy $E(S)$ consumed by S, and then divide by the schedule length $L(S)$:

$$P(S) = E(S)/L(S) \tag{4}$$

We compute $E(S)$ with Eq (5) below, by summing the contribution of each processor and of each communication link:

$$\begin{aligned} E(S) = &\sum_{j=1}^{|\mathcal{P}|} \left(P^p_{ind} \cdot L(S) + \sum_{o_i \in p_j} P^j_d \, \mathcal{E}xe(o_i, p_j, f_{i,j}) \right) \\ &+ \sum_{k=1}^{|\mathcal{L}|} \left(P^\ell_{ind} \cdot L(S) + \sum_{d_i \in \ell_k} P^k_d \, \mathcal{E}xe(d_i, \ell_k, f_\ell) \right) \end{aligned} \tag{5}$$

The first sum over $|\mathcal{P}|$ accounts for the processors while the second sum over $|\mathcal{L}|$ accounts for the communication links. Whether a processor is active or idle, it always consumes at least P^p_{ind} watts, hence the first term $P^p_{ind} \cdot L(S)$. The second term sums the contributions of all the operations o_i executed by processor p_j: when the processor is executing o_i at the voltage $V_{i,j}$ and the frequency $f_{i,j}$, the additional energy consumed is the product of the active power P^j_d by o_i's execution time. The description is similar for the communication links. $E(S)$ can be rewritten as Eq (6):

$$\begin{aligned} E(S) = &|\mathcal{P}| \cdot P^p_{ind} \cdot L(S) + C^p_{ef} \sum_{j=1}^{|\mathcal{P}|} \left(\sum_{o_i \in p_j} V^2_{i,j} \, f_{i,j} \, \mathcal{E}xe(o_i, p_j, f_{i,j}) \right) \\ &+ |\mathcal{L}| \cdot P^\ell_{ind} \cdot L(S) + C^\ell_{ef} \sum_{k=1}^{|\mathcal{L}|} \left(\sum_{d_i \in \ell_k} V^2_\ell \, f_\ell \, \mathcal{E}xe(d_i, \ell_k, f_\ell) \right) \end{aligned} \tag{6}$$

3.6 Failure hypothesis

Both processors and communication links can fail, and they are *fail-silent* (a behavior that can be achieved at a reasonable cost [3]). Classically, we adopt the failure model of Shatz and Wang [25]: failures are *transient* and the maximal duration of a failure is such that it affects only the current operation executing onto the faulty processor, and not the subsequent operations (same for the communication links); this is the "hot"

failure model. The occurrence of failures on a processor (same for a communication link) follows a Poisson law with a constant parameter λ, called its *failure rate per time unit*. Modern fail-silent processors can have a failure rate around $10^{-6}/hr$ [3].

Failures are *transient*. Those are the most common failures in modern embedded systems, all the more when processor voltage is lowered to reduce the energy consumption, because even very low energy particles are likely to create a critical charge leading to a transient failure [30]. Besides, failure occurrences are assumed to be *statistically independent events*. For hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [16].

The *reliability* of a system is defined as the probability that it operates correctly during a given time interval [1]. According to our model, the reliability of the processor P (resp. the communication link L) during the duration d is $R = e^{-\lambda d}$. Conversely, the *probability of failure* of the processor P (resp. the communication link L) during the duration d is $F = 1 - R = 1 - e^{-\lambda d}$. Hence, the reliability of the operation or data-dependency X placed onto the hardware component C (be it a processor or a communication link) is:

$$R(X, C) = e^{-\lambda_C \, \mathcal{E}xe(X,C,f)} \tag{7}$$

From now on, the function R will either be used with two variables as in Eq (7), or with only one variable to denote the reliability of a schedule (or a part of a schedule).

Since the architecture is homogeneous, the failure rate per time unit is identical for each processor (noted λ_p) and similarly for each communication link (noted λ_ℓ).

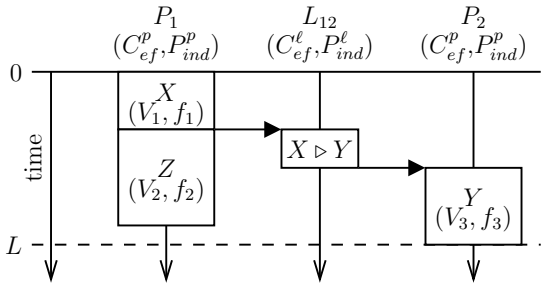


Fig. 6 A simple schedule of length L.

Figure 6 shows a simple schedule S where operations X and Z are placed onto P_1, operation Y onto processor P_2, and the data-dependency $X \triangleright Y$ is placed onto the link L_{12}. We detail below the contribution of each hardware component to the consumed energy according to Eq (6):

- On P_1: $E(P_1) = P^p_{ind} \cdot L + C^p_{ef} \left(V_1^2 \, f_1 \, \mathcal{E}xe(X, P_1, f_1) + V_2^2 \, f_2 \, \mathcal{E}xe(Z, P_1, f_2) \right)$.
- On P_2: $E(P_2) = P^p_{ind} \cdot L + C^p_{ef} \, V_3^2 \, f_3 \, \mathcal{E}xe(Y, P_2, f_3)$.
- On L_{12}: $E(L_{12}) = P^\ell_{ind} \cdot L + C^\ell_{ef} \, V_\ell^2 \, f_\ell \, \mathcal{E}xe(X \triangleright Y, L_{12}, f_\ell)$.

3.7 Summary of the assumptions

1. The algorithm graph $\mathcal{A}lg$ is a single clocked synchronous data flow graph. This is reasonable for automatic control software found in numerous embedded systems. Such systems are commonly called "periodic sampled systems".
2. The distributed memory architecture $\mathcal{A}rc$ is homogeneous and fully connected. This is reasonable since more and more embedded processors are many-core ones.

3. The switch capacitances C^p_{ef} and C^ℓ_{ef} are constant, respectively for all processors and all operations, and for all communication links and all data dependencies. This is a simplification since the switch capacitance depends on the operations actually executed by the processor. However, for the size of applications we consider, taking an average constant value is reasonable.
4. Hardware elements are fail-silent. Fail silence can be achieved at a reasonable cost, for instance with a dual lock-step processor [3].
5. Failures are transient and their duration is such that it affects only the current operation executing onto the faulty processor, and not the subsequent operations (same for the communication links). Single event upsets (SEUs), which are the most common failures affecting hardware elements, fall in this category.
6. Failure occurrences are statistically independent events. For hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [16].
7. The occurrence of failures on a hardware element follows a Poisson law with a constant parameter λ. Over the life, λ changes according to a "bathtub" curve, with a "flat" portion in the middle. Thanks to this flat portion, a constant λ can be reasonably assumed for the processors usually deployed in safety critical systems.

4 The tricriteria scheduling algorithm TSH

4.1 Global system failure rate (GSFR)

As we have demonstrated in Section 2, we must use the *global system failure rate (GSFR)* instead of the system's reliability as a criterion. The GSFR is the failure rate per time unit of the obtained multiprocessor schedule, seen as if it were a *single* operation scheduled onto a *single* processor [12]. The GSFR of a static schedule S, noted $\Lambda(S)$, is computed by Eq (8):

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} \tag{8}$$

Eq (8) uses the reliability $R(S)$, which, in the case of a static schedule S without replication, is simply the product of the reliability of each operation and data dependency of S (by definition of the reliability, Section 3.6):

$$R(S) = \prod_{(o_i,p_j)\in S} R(o_i,p_j) \cdot \prod_{(d_i,\ell_k)\in S} R(d_i,\ell_k) \tag{9}$$

Eq (8) also uses the total processor utilization $U(S)$ instead of the schedule length $L(S)$, so that the GSFR can be computed *compositionally*:

$$U(S) = \sum_{(o_i,p_j)\in S} \mathcal{E}xe(o_i,p_j,f_{i,j}) + \sum_{(d_i,\ell_k)\in S} \mathcal{E}xe(d_i,\ell_k,f_\ell) \tag{10}$$

Thanks to Eqs (8), (9), and (10), the GSFR is *invariant*: for any schedules S_1 and S_2 such that $S = S_1 \circ S_2$, where "$\circ$" is the concatenation of schedules, if $\Lambda(S_1) \leq K$ and $\Lambda(S_2) \leq K$, then $\Lambda(S) \leq K$ (Proposition 5.1 in [12]).

Finally, it is very easy to translate a reliability objective R_{obj} into a GSFR objective Λ_{obj}: one just needs to apply the formula $\Lambda_{obj} = -\log R_{obj}/D$, where D is the mission duration. This shows how to use the GSFR criterion in practice.

4.2 Decreasing the power consumption

Two operation parameters of a chip can be modified to lower the power consumption: the frequency and the voltage. We assume that each processor can be operated with a *finite set of supply voltages*, noted $\mathcal{V}$. We thus have $\mathcal{V} = \{V_0, V_1, \ldots, V_{max}\}$. To each supply voltage V corresponds an operating frequency f. We choose not to modify the operating frequency and the supply voltage of the communication links.

We assume that the cache size is adapted to the application, therefore ensuring that the execution time of an application is linearly related to the frequency [22] (i.e., the execution time is doubled when frequency is halved).

To lower the energy consumption of a chip, we use *Dynamic Voltage and Frequency Scaling (DVFS)*, which lowers the voltage and increases proportionally the cycle period. However, DVFS has an impact of the failure rate [30]. Indeed, lower voltage leads to smaller critical energy, hence the system becomes sensitive to lower energy particles. As a result, the fault probability increases both due to the longer execution time and to the lower energy: the voltage-dependent failure rate $\lambda(f)$ is:

$$\lambda(f) = \lambda_0 \cdot 10^{\frac{b(1-f)}{1-f_{min}}} \tag{11}$$

where λ_0 is the nominal failure rate per time unit, $b > 0$ is a constant, f is the frequency scaling factor, and f_{min} is the lowest operating frequency. At f_{min} and V_{min}, the failure rate is maximal: $\lambda_{max} = \lambda(f_{min}) = \lambda_0 \cdot 10^b$.

We apply DVFS to the processors and we assume that the voltage switch time can be neglected compared to the WCET of the operations. To take into account the voltage in the schedule, we modify the spatial allocation function Ω_O to give the supply voltage of the processor for each operation: $\Omega_O : \mathcal{X} \to \mathcal{Q}$, where $\mathcal{Q}$ is the domain of the sets of pairs $\langle p, v \rangle \in \mathcal{P} \times \mathcal{V}$.

To compute the number of elements in $\mathcal{Q}$, we count the number of sets of pairs $\langle p, v \rangle$ for each element of $2^{\mathcal{P}}$ except the empty set. Each element $E \in 2^{\mathcal{P}}$ accounts for $|\mathcal{V}|^{|E|}$ elements in $\mathcal{Q}$. Take for example $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ and $\mathcal{V} = \{v_1, v_2, v_3\}$. Each of the 4 singletons of $2^{\mathcal{P}}$ accounts for $3^1 = 3$ elements, each of the 6 doubletons accounts for $3^2 = 9$ elements, each of the 4 triplets accounts for $3^3 = 27$ elements, and finally the only quadruplet accounts for $3^4 = 81$ elements. The total is therefore $255 = 4^4 - 1$ elements. For instance, $\{\langle p_1, v_2 \rangle, \langle p_3, v_3 \rangle, \langle p_4, v_1 \rangle\}$ is a triplet of $\mathcal{Q}$, meaning that the concerned operation o is replicated three times, namely on p_1 at voltage v_2, on p_3 at voltage v_3, and on p_4 at voltage v_1. In the general case, it can be shown that $|\mathcal{Q}| = (|\mathcal{V}| + 1)^{|\mathcal{P}|} - 1$.

4.3 Decreasing the GSFR

According to Eq (8), *decreasing* the GSFR is equivalent to *increasing* the reliability. Several techniques can be used to increase the reliability of a system. Their common point is to include some form of redundancy (this is because the target architecture $\mathcal{A}rc$, with the failure rates of its components, is fixed) [11]. We have chosen the *active replication* of the operations and the data-dependencies, which consists in executing several copies of a same operation onto as many distinct processors (resp. data-dependencies onto communication links). Adding more replicas increases the reliability, but also, in general, the schedule length: in this sense, we say that the two criteria, length and reliability, are *antagonistic*.

To compute the GSFR of a static schedule with replication, we use *Reliability Block-Diagrams (RBD)* [2,20]. An RBD is an *acyclic oriented graph* (N, E), where

each node of N is a *block* representing an element of the system, and each arc of E is a *causality link* between two blocks. Two particular connection points are its *source* S and its *destination* D. An RBD is *operational* if and only if there exists at least one operational path from S to D. A path is operational if and only if all the blocks in this path are operational. The probability that a block be operational is its reliability. By construction, the probability that an RBD be operational is thus the reliability of the system it represents.

In our case, the system is the multiprocessor static schedule, possibly partial, of $\mathcal{A}lg$ onto $\mathcal{A}rc$. Each block represents an operation X placed onto a processor P_i or a data-dependency $X \triangleright Y$ placed onto a communication link L_j. The reliability of a block is therefore computed according to Eq (7).

Computing the reliability in this way requires the occurrences of the failures to be *statistically independent events*. Without this hypothesis, the fact that some blocks belong to several paths from S to D makes the reliability computation infeasible. At each iteration of the scheduling heuristic, we compute the RBD of the partial schedule obtained so far, then we compute the reliability based on this RBD, and finally we compute the GSFR of the partial schedule with Eq (8).

Finally, computing the reliability of an RBD with replications is, in general, exponential in the size of the schedule. To avoid this problem, we insert *routing operations* so that the RBD of any partial schedule is always *serial-parallel* (i.e., a sequence of parallel macro-blocks), hence making the GSFR computation *linear* [12]. The idea is that, for each data dependency $X \triangleright Y$ such that it has been decided to replicate X k times and Y ℓ times, a routing operation R will collect all the data sent by the k replicas of X and send it to the ℓ replicas of Y (see Figure 7).

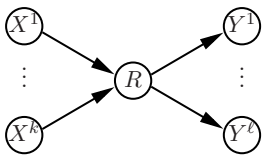


Fig. 7 A routing operation between k replicas of X and ℓ replicas of Y.

This scheme, known as "replication for reliability" [13], has a drawback in terms of schedule length, because the routing operation R cannot complete before it has received the data sent by *all* the replicas of X. However, it has been shown in [12] that, on average, the overhead of inserting routing operations on the schedule length is less than 4%.

4.4 Principle of the scheduling heuristic TSH

To obtain the Pareto front in the space (length,GSFR,power), we predefine a virtual grid in the objective plane (GSFR,power), and for each cell of the grid we solve one different single objective problem constrained to this cell, by using the scheduling heuristic TSH presented below. The single objective is the schedule length that TSH aims at minimizing.

TSH is a *ready list scheduling heuristic*. It takes as input an algorithm graph $\mathcal{A}lg$, a homogeneous architecture graph $\mathcal{A}rc$, the function $\mathcal{E}xe$ giving the WCETs and WCCTs, and two constraints Λ_{obj} and P_{obj}. It produces as output a static multiprocessor schedule S of $\mathcal{A}lg$ onto $\mathcal{A}rc$, such that the GSFR of S is smaller than Λ_{obj}, the power consumption is smaller than P_{obj}, and such that its length is as small as possible. TSH uses

active replication of operations to meet the Λ_{obj} constraint, *dynamic voltage scaling* to meet the P_{obj} constraint, and the *power-efficient schedule pressure* as a cost function to minimize the schedule length.

Besides, TSH inserts *routing operations* to make sure that the RBD of any partial schedule is serial-parallel (otherwise, computing the reliability is exponential in the size of the schedule – see Section 4.3).

TSH works with two lists of operations of $\mathcal{A}lg$: the ready operations $\mathcal{O}^{(n)}_{ready}$ and the previously scheduled operations $\mathcal{O}^{(n)}_{sched}$. The superscript (n) denotes the current iteration of the scheduling algorithm. One operation is scheduled at each iteration (n). Initially, $\mathcal{O}^{(0)}_{sched}$ is empty while $\mathcal{O}^{(0)}_{ready}$ contains the input operations of $\mathcal{A}lg$, i.e., all the operations without any predecessor. At any iteration (n), all the operations in $\mathcal{O}^{(n)}_{ready}$ are such that all their predecessors are in $\mathcal{O}^{(n)}_{sched}$. For the ease of notation, we sometimes write $P^{(n)}$ for $P\big(S^{(n)}\big)$, and similarly for the schedule length L, the energy E, or the GSFR Λ.

4.5 power-efficient schedule pressure

The power-efficient schedule pressure is a variant of the schedule pressure cost function [14], which tries to minimize the length of the critical path of the algorithm graph by exploiting the scheduling margin of each operation. The *schedule pressure* σ is computed for each ready operation o_i and each processor p_j by Eq (12):

$$\sigma^{(n)}(o_i, p_j) = ETS^{(n)}(o_i, p_j) + LTE^{(n)}(o_i) - CPL^{(n)} \tag{12}$$

where $CPL^{(n)}$ is the critical path length of the partial schedule at step (n) composed of the already scheduled operations, $ETS^{(n)}(o_i, p_j)$ is the earliest time at which the operation o_i can start its execution on the processor p_j, and $LTE^{(n)}(o_i)$ is the latest start time from end of o_i, defined to be the length of the longest path from o_i to $\mathcal{A}lg$'s output operations; this path contains the "future" operations of o_i. When computing $LTE^{(n)}(o_i)$, since the future operations of o_i are not scheduled yet, we do not know their actual voltage, and therefore neither what their execution time will be (this will only be known when these future operations will be actually scheduled). Hence, for each future operation, we compute its average WCET for all existing supply voltages.

Eq (13) generalizes the schedule pressure to a set of processors:

$$\begin{aligned} \sigma^{(n)}(o_i, \mathcal{P}_k) &= ETS^{(n)}(o_i, \mathcal{P}_k) + LTE^{(n)}(o_i) - CPL^{(n)} \\ ETS^{(n)}(o_i, \mathcal{P}_k) &= \max_{p_j \in \mathcal{P}_k} ETS^{(n)}(o_i, p_j) \end{aligned} \tag{13}$$

Then, we consider the schedule length as a criterion to be minimized, and the energy increase and the GSFR as two constraints to be met: for each ready operation $o_i \in \mathcal{O}^{(n)}_{ready}$, the *power-efficient schedule pressure* of Eq (14) computes the best subset $\mathcal{Q}^{(n)}_{best}(o_i)$ of pairs $\langle$processor, voltage$\rangle$ to execute o_i:

$$\begin{aligned} &\mathcal{Q}^{(n)}_{best}(o_i) = \mathcal{Q}_j \text{ such that:} \\ &\sigma^{(n)}(o_i, \mathcal{Q}_j) = \min_{\mathcal{Q}_k \in \mathcal{Q}} \Big\{ \sigma^{(n)}(o_i, \mathcal{Q}_k) \;\mid\; \big(E^{(n+1)} - E^{(n)}\big) \le P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \\ &\qquad\qquad \wedge\, \Lambda_B(o_i, \mathcal{Q}_k) \le \Lambda_{obj} \Big\} \end{aligned} \tag{14}$$

where $\mathcal{Q}$ is the set of all subsets of pairs $\langle p, v\rangle$ such that $p \in \mathcal{P}$ and $v \in \mathcal{V}$ (see Section 4.2), $\Lambda_B(o_i, \mathcal{Q}_k)$ is the GSFR of the parallel macro-block that contains the replicas of o_i on all the processors of $\mathcal{Q}_k$[1], and $\big(E^{(n+1)} - E^{(n)}\big)$ is the energy increase due to the scheduling of o_i on all the processors of $\mathcal{Q}_k$ at their respective voltages, where $E^{(n+1)}$ and $E^{(n)}$ are computed by Eq (6), $E^{(n)}$ begin the energy before scheduling o_i and $E^{(n+1)}$ after. Finally, when computing $\Lambda_B(o_i, \mathcal{Q}_k)$, the failure rate of each processor is computed by Eq (11) according to its voltage in $\mathcal{Q}_k$. Finally, the constraint on the energy consumption $\big(E^{(n+1)} - E^{(n)}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big)$ is equivalent to a constraint on the power consumption, $\big(E^{(n+1)} - E^{(n)}\big)/\big(L^{(n+1)} - L^{(n)}\big) \leq P_{obj}$, but without the potential division by zero.

The local constraint on the current macro-block of the RBD, $\Lambda_B(o_i, \mathcal{Q}_k) \leq \Lambda_{obj}$, guarantees that the global constraint on the schedule at iteration $(n+1)$, $\Lambda^{(n+1)} \leq \Lambda_{obj}$, is met. This will be formally established by Proposition 1 (see Section 4.7).

Similarly, we would like the local constraint on the energy increase due to o_i, $\big(E^{(n+1)} - E^{(n)}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big)$, to guarantee that the global constraint at iteration $(n+1)$ on the full schedule $P^{(n+1)} \leq P_{obj}$ is met. Unfortunately, we can show a counter example for this.

Consider the case when the operation o_i scheduled at iteration (n) *does not* increase the schedule length, because it fits in a slack at the end of the previous schedule: $L^{(n+1)} = L^{(n)}$. In contrast, the total energy always increases *strictly* because of o_i: $\big(E^{(n+1)} - E^{(n)}\big) > 0$. It follows that, whatever the choice of processors, voltage, and frequency for o_i, it is *impossible* to schedule it such that the energy increase constraint $\big(E^{(n+1)} - E^{(n)}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big)$ be met.

4.6 Over-estimation of the energy consumption

To prevent this and to guarantee the invariance property of P, we *over-estimate* the power consumption, by computing the consumed energy as if all the ending slacks were "filled" by an operation executed at (f_{over}, V_{over}). We choose the largest frequency and voltage (f_{over}, V_{over}) such that:

$$\big(E^{(n+1)} - E^{(n)}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \tag{15}$$

We start with (f_{max}, V_{max}). If the Condition (15) is not met, then we select the next highest operating frequency, and so on until Condition (15) is met. Thanks to this over-estimation, even if the next scheduled operation fits in a slack and does not increase the length, we are sure that it will not increase the power-consumption either. This is illustrated in Figure 8.

Formally, we now compute the total energy consumed by the schedule S with Eq (16) instead of Eq (6). We call E_+ the *over-estimated energy consumption*:

$$\begin{aligned} E_+(S) = |\mathcal{P}|\cdot P^p_{ind}\cdot L(S) + C^p_{ef}\sum_{j=1}^{|\mathcal{P}|}\left(\sum_{o_i\in p_j} V^2_{i,j}\, f_{i,j}\, \mathcal{E}xe(o_i, p_j, f_{i,j}) + V^2_{over}\big(L(S) - M_j\big)\right) \\ + |\mathcal{L}|\cdot P^\ell_{ind}\cdot L(S) + C^\ell_{ef}\sum_{k=1}^{|\mathcal{L}|}\left(\sum_{d_i\in\ell_k} V^2_\ell\, f_\ell\, \mathcal{E}xe(d_i, \ell_k, f_\ell)\right) \end{aligned} \tag{16}$$

[1] Because $\Lambda_B(o_i, \mathcal{Q}_k)$ is not the GSFR of the partial schedule $S^{(n+1)}$ but only of the macro-block of o_i, it does not bear the superscript $(n+1)$.

where, $L(S) - M_j$ is the slack available at the end of processor p_j, for all processor $p_j \in \mathcal{P}$, $M_j = \max_{o_i \in p_j} \big(st(o_i, p_j) + \mathcal{E}xe(o_i, p_j, f_{i,j})\big)$ is the completion time of the last operation scheduled on p_j, and for all operation o_i scheduled on p_j, $st(o_i, p_j)$ is the start time of o_i on p_j. Compared to Eq (6), we see that the over-estimating term $V^2_{over}\big(L(S) - M_j\big)$ has been added.

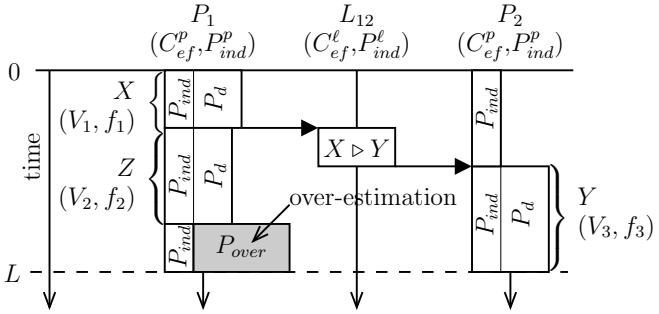


Fig. 8 Over-estimation of the energy consumption.

Accordingly, we now compute the power-efficient schedule pressure with Eq (17) instead of Eq (14):

$$\begin{aligned}\mathcal{Q}^{(n)}_{best}(o_i) = \mathcal{Q}_j \text{ such that:}\\ \sigma^{(n)}(o_i, \mathcal{Q}_j) = \min_{\mathcal{Q}_k \in \mathcal{Q}} \Big\{ \sigma^{(n)}(o_i, \mathcal{Q}_k) \;\mid\; \big(E^{(n+1)}_{+} - E^{(n)}_{+}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \\ \wedge\, \Lambda_B(o_i, \mathcal{Q}_k) \leq \Lambda_{obj} \Big\}\end{aligned} \tag{17}$$

Once we have computed, for each ready operation o_i of $\mathcal{O}^{(n)}_{ready}$, the best subset of pairs $\langle$processor, voltage$\rangle$ to execute o_i, with the power-efficient schedule pressure of Eq (17), we compute the *most urgent* of these operations with Eq (18):

$$o_{urg} = o_i \in \mathcal{O}^{(n)}_{ready} \text{ s.t. } \sigma^{(n)}\big(o_i, \mathcal{Q}^{(n)}_{best}(o_i)\big) = \max_{o_j \in \mathcal{O}^{(n)}_{ready}} \Big\{ \sigma^{(n)}\big(o_j, \mathcal{Q}^{(n)}_{best}(o_j)\big) \Big\} \tag{18}$$

Finally, we schedule this most urgent operation o_{urg} on the processors of the set $\mathcal{Q}^{(n)}_{best}(o_{urg})$, and we end the current iteration (n) by updating the lists of scheduled and ready operations. Firstly, the newly scheduled operation o_{urg} is added to the list of scheduled operations: $\mathcal{O}^{(n+1)}_{sched} := \mathcal{O}^{(n)}_{sched} \cup \{o_{urg}\}$. Secondly, o_{urg} is removed from the list of ready operations, which is also augmented with all the successors of o_{urg} that happen to have all their predecessors in the new list of scheduled operations: $\mathcal{O}^{(n+1)}_{ready} := \mathcal{O}^{(n)}_{ready} - \{o_{urg}\} \cup \{o' \in succ(o_{urg}) \mid pred(o') \subseteq \mathcal{O}^{(n+1)}_{sched}\}$.

4.7 Soundness of TSH

The soundness of TSH is is based on two propositions. The first one establishes that the schedules produced by TSH meet their GSFR constraint. Its proof can be found in [12]:

Proposition 1 *Let S be a multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. If each operation o of $\mathcal{A}lg$ has been scheduled according to Eqs (17) and (18) such that the reliability is computed with Eq (8), then the GSFR $\Lambda(S)$ is less than Λ_{obj}.*

The second proposition establishes that the schedules produced by TSH meet their power consumption constraint:

Proposition 2 *Let S be a multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. If each operation o of $\mathcal{A}lg$ has been scheduled according to Eqs (17) and (18) such that the energy consumption is computed with Eq (16), then the total power consumption $P(S)$ is less than P_{obj}.*

Proof First, we observe that, for any non empty schedule S, $P(S) \leq P_{obj}$ is equivalent to $E(S) \leq P_{obj}\, L(S)$. Moreover, since $E(S) \leq E_+(S)$, it is sufficient to prove the inequality of Eq (19):

$$E_+(S) \leq P_{obj}\, L(S) \tag{19}$$

We prove Eq (19) by induction on the scheduling iteration (n). The induction hypothesis $[H]$ is on the energy consumed by the partial schedule $S^{(n)}$:

$$E_+^{(n)} \leq P_{obj}\, L^{(n)} \qquad\qquad [H]$$

$[H]$ is satisfied for the initial empty schedule $S^{(0)}$ because $E_+^{(0)} = 0$ and $L^{(0)} = 0$. Let o_{urg} be the operation chosen by Eq (18) at iteration (n) to be scheduled on all the processors of the set $\mathcal{Q}_{best}^{(n)}(o_{urg})$. According to Eq (17), we have:

$$\begin{aligned} & \big(E_+^{(n+1)} - E_+^{(n)}\big) \leq P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \\ \iff\ & E_+^{(n+1)} \leq E_+^{(n)} + P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \end{aligned}$$

Thanks to [H], we thus have:

$$\begin{aligned} & E_+^{(n+1)} \leq P_{obj}\, L^{(n)} + P_{obj}\big(L^{(n+1)} - L^{(n)}\big) \\ \iff\ & E_+^{(n+1)} \leq P_{obj}\, L^{(n+1)} \end{aligned}$$

As a conclusion, $[H]$ holds for the schedule $S^{(n+1)}$. □

4.8 The TSH algorithm

The TSH scheduling heuristic is shown in Figure 9. Initially, $\mathcal{O}_{sched}^{(0)}$ is empty and $\mathcal{O}_{ready}^{(0)}$ is the list of operations without any predecessors. At the end of each iteration (n), these lists are updated according to the data-dependencies of $\mathcal{A}lg$.

At each iteration (n), one operation o_i of the list $\mathcal{O}_{ready}^{(n)}$ is selected to be scheduled. For this, we select at the micro-steps ① and ②, for each ready operation o_i, the best subset of processors $\mathcal{Q}_{best}^{(n)}(o_i)$ to replicate and schedule o_i, such that the GSFR of the resulting partial schedule is less than Λ_{obj} and the power consumption is less than P_{obj}; at this point, each processor is selected with a voltage. Then, among those best pairs $\langle o_i, \mathcal{Q}_{best}^{(n)}(o_i)\rangle$, we select at the micro-step ③ the one having the biggest power-efficient schedule pressure value, i.e., the most urgent pair $\langle o_{urg}, \mathcal{Q}_{best}^{(n)}(o_{urg})\rangle$.

In Section 6, we will present a complete set of simulation results, first involving TSH alone, then comparing TSH with a multicriteria heuristic from the literature, and finally comparing TSH with an optimal Mixed Linear Integer Program.

Algorithm TSH:
input: $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{E}xe$, Λ_{obj}, and P_{obj};
output: a multi-processor static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$ that minimizes the length and satisfies Λ_{obj} and P_{obj}, or a failure message;
begin
Compute the set $\mathcal{Q}$ of all subsets of pairs $\langle$processor, voltage$\rangle$;
$\mathcal{O}_{ready}^{(0)} := \{$operations without predecessors$\}$;
$\mathcal{O}_{sched}^{(0)} := \emptyset$;
$n := 0$;
while $(\mathcal{O}_{ready}^{(n)} \neq \emptyset)$ **do**

① For each ready operation $o_i \in \mathcal{O}_{ready}^{(n)}$, compute $\sigma^{(n)}(o_i, \mathcal{Q}_k)$ for each $\mathcal{Q}_k \in \mathcal{Q}$.

② For each ready operation o_i, select the best subset $\mathcal{Q}_{best}^{(n)}(o_i) \in \mathcal{Q}$ such that:

$$\sigma^{(n)}(o_i, \mathcal{Q}_j) = \min_{\mathcal{Q}_k \in \mathcal{Q}} \left\{ \sigma^{(n)}(o_i, \mathcal{Q}_k) \;\mid\; \left(E_+^{(n+1)} - E_+^{(n)}\right) \leq P_{obj}\left(L^{(n+1)} - L^{(n)}\right) \wedge \Lambda_B(o_i, \mathcal{Q}_k) \leq \Lambda_{obj} \right\}$$

③ Select the most urgent ready operation o_{urg} among all o_i of $\mathcal{O}_{ready}^{(n)}$ such that:

$$\sigma^{(n)}\left(o_{urg}, \mathcal{Q}_{best}^{(n)}(o_{urg})\right) = \max_{o_j \in \mathcal{O}_{ready}^{(n)}} \left\{ \sigma^{(n)}\left(o_j, \mathcal{Q}_{best}^{(n)}(o_j)\right) \right\}$$

④ Schedule each replica of o_{urg} on each processor of $\mathcal{Q}_{best}^{(n)}(o_{urg})$ and at the voltage specified in $\mathcal{Q}_{best}^{(n)}(o_{urg})$;

⑤ **if** $(\mathcal{Q}^{(n+1)}(o_{urg}) = \emptyset)$ **then**
return "fail to satisfy the constraints"; **exit**;
/* the user can modify Λ_{obj} or P_{obj} and re-run TSH */

⑥ Update the lists of ready and scheduled operations:
$\mathcal{O}_{sched}^{(n+1)} := \mathcal{O}_{sched}^{(n)} \cup \{o_{urg}\}$;
$\mathcal{O}_{ready}^{(n+1)} := \mathcal{O}_{ready}^{(n)} - \{o_{urg}\} \cup \{o' \in succ(o_{urg}) \mid pred(o') \subseteq \mathcal{O}_{sched}^{(n+1)}\}$;

⑦ $n := n + 1$;

end while
end

Fig. 9 The TSH tricriteria scheduling heuristic.

5 Mixed Integer Linear Programming approach

In this section, we define a mixed integer linear programming model (MILP) for our scheduling problem. Our goal is to compare the optimal results obtained by our MILP program with those achieved by TSH on small $\mathcal{A}lg$ graphs. Comparisons will be shown in Section 6.4.

For each operation t_i of $\mathcal{A}lg$, let $s_{ik} \in \mathbb{R}^+$ be the starting execution time of its k-th replica:

$$\forall i,\ s_{ik} \geq 0 \tag{20}$$

Let $p_{ik} \in \mathbb{N}$ be the processor index where the k-th replica of operation t_i is to be executed. The value 0 indicates that no processor is selected:

$$\forall i,\ p_{ik} \in \{0, \ldots, |\mathcal{P}|\} \tag{21}$$

Let $x_{ik\ell}$ be 1 if the k-th replica of operation t_i is assigned to processor number ℓ, and 0 otherwise:

$$\forall i,\ \forall k,\ \forall \ell,\ x_{ik\ell} \in \{0,1\} \tag{22}$$

$$\forall i,\ \forall k,\ \sum_{\ell} x_{ik\ell} = 1 \tag{23}$$

The constraints (24) link the mapping variables x with the processors indices p:

$$\forall i,\ \forall k,\ p_{ik} = \sum_{\ell} \ell\, x_{ik\ell} \tag{24}$$

Let $x_{ik\ell m}$ be 1 if the k-th replica of operation t_i is assigned to processor number ℓ and runs with frequency m, and 0 otherwise:

$$\forall i,\ \forall k,\ \forall \ell,\ \forall m,\ x_{ik\ell m} \in \{0,1\} \tag{25}$$

$$\forall i,\ \forall k,\ \forall \ell,\ \sum_{m} x_{ik\ell m} = x_{ik\ell} \tag{26}$$

We can then define W as the schedule length of $\mathcal{A}lg$ on $\mathcal{A}rc$ (the makespan):

$$\forall i,\ \forall k,\ s_{ik} + \sum_{k,\ell,m} x_{ik\ell m}\, \mathcal{E}xe(t_i, P_\ell, f_m) \leq W \tag{27}$$

Let U be the total utilization of the processors of $\mathcal{A}rc$:

$$U = \sum_{i,k} \sum_{\ell,m} \mathcal{E}xe(t_{ik}, P_\ell, f_m) \tag{28}$$

The two global objectives Λ_{obj} and R_{obj} are related by the reliability formula:

$$\Lambda_{obj}\, U = -\log(R_{obj}) \tag{29}$$

In order to model the non-overlapping of operations and to reflect the fact that the multiprocessor schedule must enforce the precedence of the $\mathcal{A}lg$ graph, we define two sets of binary variables $\sigma_{ikjk'}$ and $\varepsilon_{ikjk'}$ such that:

– for each i, j, $\sigma_{ikjk'}$ is equal to 1 if the k-th replica of operation t_i ends before the k'-th replica of operation t_j starts, and 0 otherwise:

$$\forall i,\ \forall j,\ \forall k,\ \forall k',\ \sigma_{ikjk'} \in \{0,1\} \tag{30}$$

– for each i, j, $\varepsilon_{ikjk'}$ is equal to 1 if the index of the processor of the replica of operation t_i is strictly less that the processor index of the replica of operation t_j, and 0 otherwise:

$$\forall i,\ \forall j,\ \forall k,\ \forall k',\ \varepsilon_{ikjk'} \in \{0,1\} \tag{31}$$

These two variables σ and ε must satisfy the following constraints:

$$\forall i \neq j,\ \forall k,\ \forall k',\ s_{jk'} - s_{ik} - \sum_{\ell,m} x_{ik\ell m}\, \mathcal{E}xe(t_i, P_\ell, f_m) - U\,\sigma_{ikjk'} + U \geq 0 \quad (32)$$

$$\forall i \neq j,\ \forall k,\ \forall k',\ p_{jk'} - p_{ik} - 1 - |\mathcal{P}|\ \varepsilon_{ikjk'} + |\mathcal{P}| \geq 0 \quad (33)$$

$$\forall i \neq j,\ \forall k,\ \forall k',\ \sigma_{ikjk'} + \sigma_{jk'ik} + \varepsilon_{ikjk'} + \varepsilon_{jk'ik} \geq 1 \quad (34)$$

$$\forall i \neq j,\ \forall k,\ \forall k',\ \sigma_{ikjk'} + \sigma_{jk'ik} \leq 1 \quad (35)$$

$$\forall i \neq j,\ \forall k,\ \forall k',\ \varepsilon_{ikjk'} + \varepsilon_{jk'ik} \leq 1 \quad (36)$$

$$\forall i \in pred(j),\ \sigma_{ikjk'} = 1 \quad (37)$$

We define the time order on operations in terms of the σ variables in (32), and similarly we define the processors indices order on operations in terms of the ε variables in (33) where $|\mathcal{P}|$ is the number of processors in $\mathcal{A}rc$. By (34), we ensure that operations do not overlap on a processor. By (35), we ensure that an operation cannot be scheduled both before and after another operation. Similarly, by (36), an operation cannot be placed both on a higher and on a lower processor index than another operation. Finally, (37) enforces the task precedence constraints.

Let Y_{iK} be a binary variable equal to 1 if the replication level for operation t_i is K, with $1 \leq K \leq R_{max}$, and 0 otherwise. Here, R_{max} is the maximal allowed replication level for the operations:

$$\forall i,\ \sum_K Y_{iK} = 1 \quad (38)$$

$$\forall i,\ \sum_{k \leq R_{max}-1} \sum_\ell x_{ik\ell} = \sum_{K \leq R_{max}} K\, Y_{iK} \quad (39)$$

We constrain the power and the reliability of $\mathcal{A}lg$ on $\mathcal{A}rc$ in (40) and (41) respectively. Here, RB_{iK} is the reliability of the operation i when replicated exactly K times on processors identified by the set L composed of K processor indices, with frequencies identified by the set M composed of K frequency values:

$$\sum_i \sum_{k,\ell,m} x_{ik\ell m}\, \mathcal{E}xe(t_{ik}, P_\ell, f_m)\, f_m^3 \leq P_{obj}\, W \quad (40)$$

$$\sum_i \sum_K Y_{iK} \sum_{L,M} \Big((\pi_{k \leq K}\, x_{ikL(k)M(k)})\ \log(RB_{iK}) \Big) \geq \log(R_{obj}) \quad (41)$$

Based on these definitions, the formulation of the MILP is to minimize the execution length W under the constraints specified by Eqs (20) to (41). This formulation is a bilinear programming where the bilinearities arise because of the reliability constraints. We have linearized this model by simply introducing a new set of variables which replace the bilinear terms.

In Section 6.4, we compare, on a given instance, the schedules obtained with this MILP and with TSH.

6 Simulation results

6.1 Examples of Pareto fronts produced by TSH

The aim of our first simulations is to produce Pareto fronts. Figures 10 and 11 show the Pareto fronts produced by TSH for a randomly generated $\mathcal{A}lg$ graph of 30 operations, and a fully connected and homogeneous $\mathcal{A}rc$ graph of respectively 3 and 4 processors;

we have used the same random graph generator as in [12]. The nominal failure rate per time unit (i.e., the λ_0 of Eq (11)) of all the processors is $\lambda_p = 10^{-5}$; the nominal failure rate per time unit of all the links is $\lambda_\ell = 5.10^{-4}$; the set of supply voltages is $\mathcal{V} = \{0.25, 0.50, 0.75, 1.0\}$ (scaling factor).

The virtual grid of the Pareto front is defined such that both high and small values of P_{obj} and Λ_{obj} are covered within a reasonable grid size. Hence, the values of P_{obj} and Λ_{obj}, from the less to the most constrained, are selected from two sets of values: $P_{obj} \in \{3.0, 2.8, 2.6, \ldots 1.0\}$ and $\Lambda_{obj} \in \{8.10^{-1}, 4.10^{-1}, 8.10^{-2}, \ldots 4.10^{-14}\}$. TSH being a heuristic, changing the parameters of this grid could change locally some points of the Pareto front, but not its overall shape.

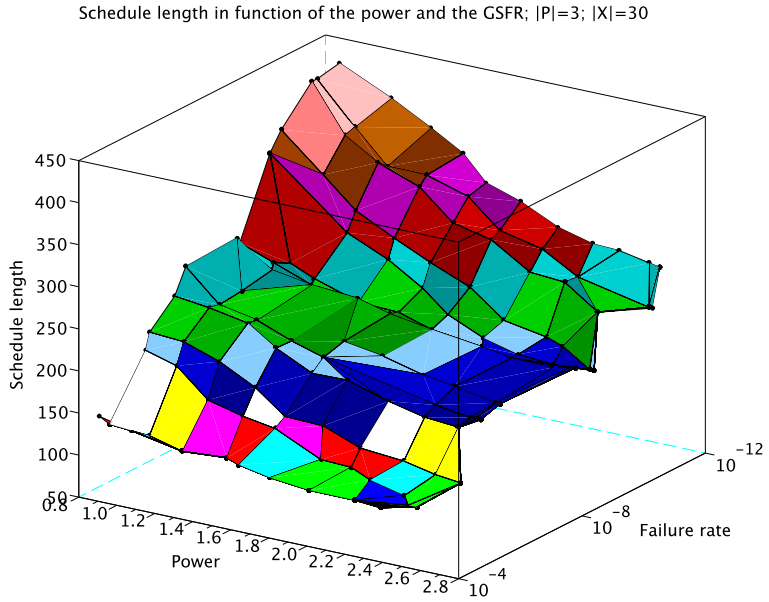


Fig. 10 Pareto front generated for an instance with 30 operations and 3 processors.

Figures 10 and 11 connect the set of non-dominated Pareto optima (the surface obtained in this way is only depicted for a better visual understanding; by no means do we assume that points interpolated in this way are themselves Pareto optima, only the computed dots are). The figures show an increase of the schedule length for points with decreasing power consumptions and/or failure rates. The "cuts" observed at the top and the left of the plots are due to low power constraints and/or low failure rates constraints.

Figures 10 and 11 exposes to the designer a choice of several tradeoffs between the execution time, the power consumption, and the reliability level. For instance, in Figure 11, we see that, to obtain a GSFR of 10^{-10} with a power consumption of 1.5 V, then we must accept a schedule three times longer than if we impose no constraint on the GSFR nor the power. We also see that, by providing a 4 processor architecture (Figure 11), we can obtain schedules with a shorter execution length than with only 3 processors, even though we impose identical constraints to the GSFR and the power

(Figure 10): with 4 processors the schedule length is in the range $[66, 374]$ time units, while with 3 processors it is in the range $[88, 437]$ time units.

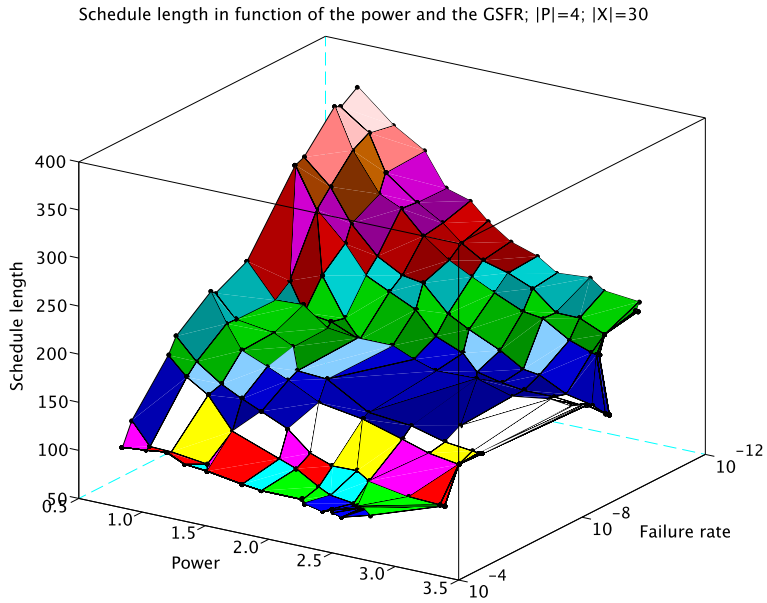
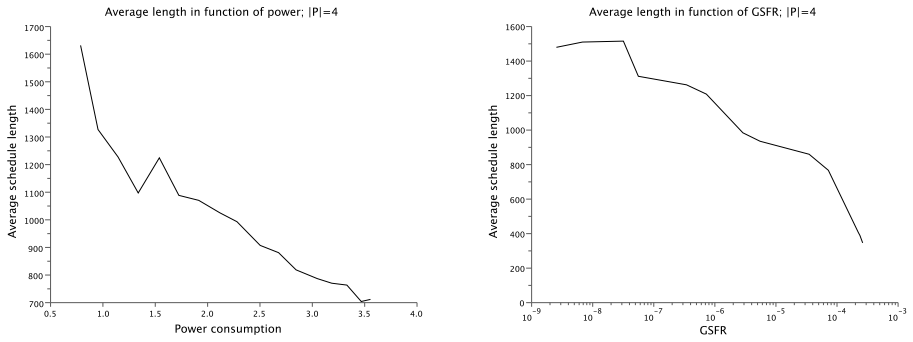


Fig. 11 Pareto front generated for an instance with 30 operations and 4 processors.

6.2 Impact of the power consumption and of the GSFR on the length

Figure 12(a) shows how the schedule length varies in function of the required power consumption, with Λ_{obj} set to 10^{-5}. This curve is averaged over 30 randomly generated $\mathcal{A}lg$ graphs. We can see that the average schedule length increases when the constraint P_{obj} on the power consumption decreases. This was expected since the two criteria, schedule length and power consumption, are antagonistic.



(a) (b)

Fig. 12 (a) Average schedule length in function of the power; (b) In function of the GSFR.

Figure 12(b) shows how the schedule length varies in function of the required GSFR, with P_{obj} set to 2.5 watt. Again, this curve is averaged over 30 randomly generated $\mathcal{A}lg$ graphs. We can see that the average schedule length increases when the constraint Λ_{obj} on the GSFR decreases. Again, this is expected because the two criteria, schedule length and GSFR, are antagonistic.

6.3 Comparison with ECS

We have compared the performance of TSH with the algorithm proposed in [18], called ECS (Energy-Conscious Scheduling heuristic). ECS is a bicriteria scheduling heuristic that takes as input a DAG of tasks and a set of p fully connected, heterogeneous, DVFS enabled, processors. The power consumption model is the same as ours, but the energy consumed by an application does not take into account the energy consumed by the inter-tasks data-dependencies on the communication links. The cost function used by ECS sums two terms, one for the energy and one for the schedule length (aggregation method). Since ECS is not *tri*criteria, we proceed as follows:

1. We first invoke ECS on a given instance (an $\mathcal{A}lg$ graph and an $\mathcal{A}rc$ graph). We then compute the overall reliability R_{ECS}, the total energy E_{ECS}, the schedule length L_{ECS}, and the total utilization U_{ECS} of the schedule produced by ECS.
2. We use these values to compute the objectives required to run TSH: $\Lambda_{obj} = -\log(R_{ECS})/U_{ECS}$ and $P_{obj} = E_{ECS}/L_{ECS}$. And finally. we invoke TSH with these values of the objectives.

We have plotted in Figures 13, 14, and 15, respectively the average schedule length, the average energy consumption, and the average reliability of the schedules computed by ECS and by TSH. The values have been averaged over 50 randomly generated $\mathcal{A}lg$ of size N varying between 10 and 100 operations. The $\mathcal{A}rc$ graph has $P = 6$ processors, and the nominal failure rate per time unit of all the processors is $\lambda_p = 10^{-5}$; the nominal failure rate per time unit of all the links is $\lambda_\ell = 5.10^{-4}$.

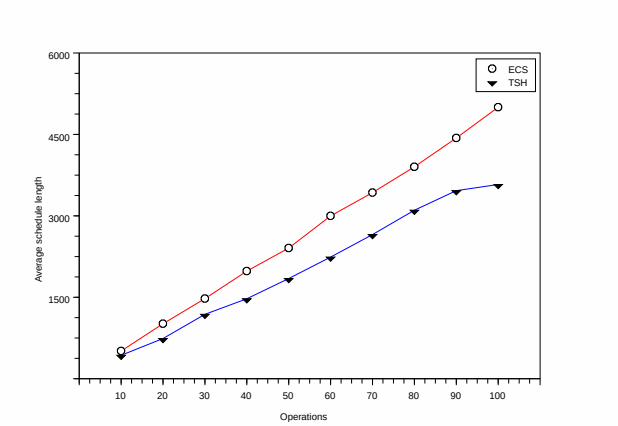


Fig. 13 Average schedule length generated by ECS and TSH.

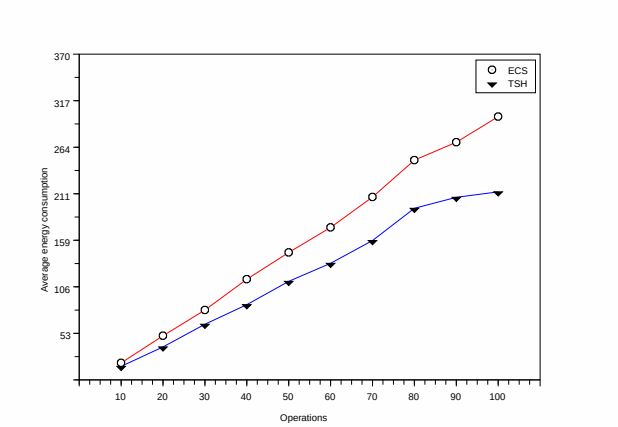


Fig. 14 Average energy consumption generated by ECS and TSH.

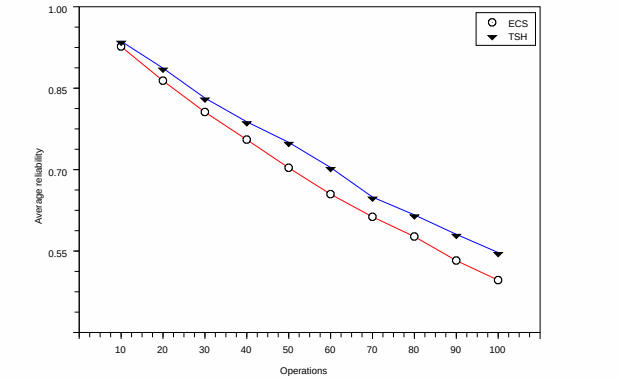


Fig. 15 Average reliability generated by ECS and TSH.

Our experimental results (Figures 13, 14 and 15) show that TSH performs systematically better than ECS. This is a very good result.

6.4 MILP and TSH simulation results

For the evaluation of MILP approach, we used an algorithm graph $\mathcal{A}lg$ of 5 operations and an architecture graph $\mathcal{A}rc$ consisting of 3 fully connected processors. The execution times of the operations were assigned randomly within 10 to 30 time units. In this simulation, we assumed that the communication links were reliable.

The nominal failure rate per time unit of the processors is $\lambda_p = 10^{-5}$. The set of processor frequencies is set to $\{0.25, 0.5, 0.75, 1\}$. The decreasing values of P_{obj} and Λ_{obj} are selected from two sets of values: $P_{obj} \in \{3.0, 2.8, 2.6, \ldots 1.0\}$ and $\Lambda_{obj} \in \{8.10^{-1}, 4.10^{-1}, 8.10^{-2}, \ldots 4.10^{-9}\}$.

We have used the CLPEX ILOG solver [7], version 11.2, on an Intel Core-2 Duo CPU E7500 2.93GHz computer with 2 GB of RAM. Even with an $\mathcal{A}lg$ graph of 7 operations, a run of MILP can take more than 40 hours without finding the optimal value. This is why we have limited the $\mathcal{A}lg$ graph to 5 operations and the $\mathcal{A}rc$ graph to 3 processors. The processing time of TSH is, as expected, much shorter than that

of the MILP: in the order of one second for TSH versus between a few seconds and 40 minutes for the MILP.

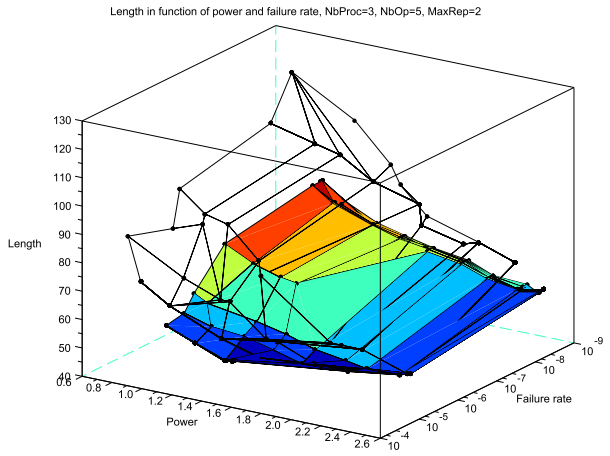


Fig. 16 Pareto fronts generated by the MILP and by TSH for an $\mathcal{A}lg$ graph of 5 operations scheduled onto an $\mathcal{A}rc$ graph of 3 processors.

The Pareto fronts generated by MILP and TSH are shown in Figure 16, where the colored surface corresponds to MILP results while the uncolored one corresponds to TSH. For small values of P_{obj} and Λ_{obj} (i.e., when the multicriteria problem is highly constrained), the TSH surface is significantly above the MILP one. For large values of P_{obj} and Λ_{obj} (i.e., when the multicriteria problem is not so constrained), the two surfaces are almost glued one to the other. The average overhead of the schedule length achieved by TSH versus the length achieved by the MILP is only 15.6 % (the exact approximation ratio is 1.1563051). This shows that TSH performs very well compared to the optimal result obtained by the MILP.

7 Related work

Many solutions exist in the literature to optimize the schedule length and the energy consumption (e.g., [23]), or to optimize the schedule length and the reliability (e.g., [8,13,4]), but very few tackle the problem of optimizing the *three* criteria (length,reliability,energy). The closest to our work are [30,24].

Zhu et al. have studied the impact of the supply voltage on the failure rate [30], in a passive redundancy framework (primary backup approach). They use DVFS to lower the energy consumption and they study the tradeoff between the energy consumption and the "performability" (defined as the probability of finishing the application correctly within its deadline in the presence of faults). A lower frequency implies a higher execution time and therefore less slack time for scheduling backup replicas, meaning a lower performability. However, their input problem is not a multiprocessor scheduling one since they study the system as a single monolithic operation executed on a single processor. Thanks to this simpler setting, they are able to provide an analytical solution based on the probability of failure, the WCET, the voltage, and the frequency.

Pop et al. have addressed the (length,reliability,energy) tricriteria optimization problem on an heterogeneous architecture [24]. Both length and reliability are taken as a constraint, respectively with a given upper and lower bound. These two criteria are *not* invariant measures, and we have demonstrated in Section 2 that such a method *cannot always guarantee* that the constraints are met. Indeed, their experimental results show that the reliability decreases with the number of processors, therefore making it impossible to meet an arbitrary reliability constraint. Secondly, they assume that the user will specify the number of processor failures to be tolerated in order to satisfy the desired reliability constraint. Thirdly, they assume that all the communications take place through a reliable bus. For these three reasons, it is not possible to compare TSH with their method.

8 Conclusion

We have presented a new off-line tricriteria scheduling heuristic, called **TSH**, which takes as input an application graph (a DAG of operations) and a multiprocessor architecture (homogeneous and fully connected), and produces a static multiprocessor schedule that optimizes three criteria: its length, its global system failure rate (GSFR), and its power consumption. TSH uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling* to lower the power consumption.

Since the three criteria of this optimization problem are *antagonistic* with each other, there is no best solution in general. This is why we use the notion of Pareto optima. To address this issue, both the power and the GSFR are taken as *constraints*, and TSH attempts to minimize the schedule length while satisfying these constraints. By running TSH with several values of these constraints, we are able to produce a set of non-dominated Pareto solutions, the Pareto front, which is a surface in the 3D space (length,GSFR,power). This surface exposes the existing tradeoffs between the three antagonistic criteria, allowing the user to choose the solution that best meets his/her application needs.

Transforming two criteria into constraints and minimizing the third criterion is a natural approach in order to produce Pareto fronts. However, some care must be taken when doing so. As we have demonstrated, each criterion that is transformed into a constraint must be an *invariant measure* of the schedule, not a varying one. For this reason, the two constraints imposed to TSH are the power consumption (instead of the energy consumption) and the global system failure rate (instead of the reliability).

TSH is an extension of our previous bicriteria (length,reliability) heuristic BSH [12]. Studying the three criteria together makes sense because of the impact of the voltage on the failure probability. Indeed, lower voltage leads to smaller critical energy, hence the system becomes sensitive to lower energy particles. As a result, the fault probability increases both due to the longer execution time and to the lower energy.

To the best of our knowledge, this is the *first* reported method that allows the user to produce the Pareto front in the 3D space (length,GSFR,power). This advance comes at the price of several assumptions: the architecture is assumed to be homogeneous and fully connected, the processors are assumed to be fail-silent and their failures are assumed to be statistically independent, the power switching time is neglected, and the failure model is assumed to be exponential. In the future, we shall work on relaxing those assumptions.

References

1. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.*, 1(1):11–33, January 2004.
2. H.S. Balaban. Some effects of redundancy on system reliability. In *National Symposium on Reliability and Quality Control*, pages 385–402, Washington (DC), USA, January 1960.
3. M. Baleani, A. Ferrari, L. Mangeruca, M. Peri, S. Pezzini, and A. Sangiovanni-Vincentelli. Fault-tolerant platforms for automotive safety-critical applications. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03*, San Jose (CA), USA, November 2003. ACM, New-York.
4. A. Benoit, F. Dufossé, A. Girault, and Y. Robert. Reliability and performance optimization of pipelined real-time systems. In *International Conference on Parallel Processing, ICPP'10*, pages 150–159, San Diego (CA), USA, September 2010.
5. T.D. Burd and R.W. Brodersen. Energy efficient CMOS micro-processor design. In *Hawaii International Conference on System Sciences, HICSS'95*, Honolulu (HI), USA, 1995. IEEE, Los Alamitos.
6. A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Syst.*, 18(2/3):249–274, 2000.
7. IBM ILOG CPLEX Optimizer. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`, 2010.
8. A. Dogan and F. Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. Parallel and Distributed Systems*, 13(3):308–323, March 2002.
9. E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Workshop on Power-Aware Computing Systems, WPACS'02*, pages 179–196, Cambridge (MA), USA, February 2002.
10. C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In *International Workshop on Embedded Software, EMSOFT'01*, volume 2211 of *LNCS*, Tahoe City (CA), USA, October 2001. Springer-Verlag.
11. F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, March 1999.
12. A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Trans. Dependable Secure Comput.*, 6(4):241–254, December 2009.
13. A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *J. of Parallel and Distributed Computing*, 69(3):326–336, March 2009.
14. T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy, May 1999. ACM, New-York.
15. T.C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9, 1961.
16. J.C. Knight and N.G. Leveson. An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Trans. Software Engin.*, 12(1):96–109, 1986.
17. H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Pub., Hingham, MA, 1997.
18. Y.C. Lee and A.Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *IEEE/ACM International Symposium on Cluster Computing and the Grid, SCCG'09*, 2009.
19. J.Y-T. Leung, editor. *Handbook of Scheduling. Algorithms: Models, and Performance Analysis*. Chapman & Hall/CRC Press, 2004.
20. D. Lloyd and M. Lipow. *Reliability: Management, Methods, and Mathematics*, chapter 9. Prentice-Hall, 1962.
21. J. Luo, L.-S. Peh, and N. Jha. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Design Automation and Test in Europe Conference, DATE'03*, pages 1150–1151, Munich, Germany, March 2003.
22. R. Melhem, D. Mossé, and E.N. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. Comput.*, 53(2):217–231, 2004.
23. T. Pering, T.D. Burd, and R.W. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *International Symposium on Low Power Electronics and Design, ISLPED'98*, pages 76–81, Monterey (CA), USA, August 1998. ACM, New-York.

24. P. Pop, K. Poulsen, and V. Izosimov. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS'07*, Salzburg, Austria, October 2007. ACM, New-York.
25. S.M. Shatz and J.-P. Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Trans. Reliability*, 38(1):16–26, April 1989.
26. J. Souyris, E.L. Pavec, G. Himbert, V. Jégu, G. Borios, and R. Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *International Workshop on Worst-case Execution Time, WCET'05*, pages 21–24, Mallorca, Spain, July 2005.
27. H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separate cache and path analyses. *Real-Time Syst.*, 18(2/3):157–179, May 2000.
28. V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag, 2006.
29. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The determination of worst-case execution times — overview of the methods and survey of tools. *ACM Trans. Embedd. Comput. Syst.*, 7(3), April 2008.
30. D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *International Conference on Computer Aided Design, ICCAD'04*, pages 35–40, San Jose (CA), USA, November 2004.