HAL
open science

Tradeoff exploration between reliability, power consumption, and execution time

Ismail Assayad, Alain Girault, Hamoudi Kalla

To cite this version:

Ismail Assayad, Alain Girault, Hamoudi Kalla. Tradeoff exploration between reliability, power consumption, and execution time. SAFECOMP, Sep 2012, Napoli, Italy. pp.437–451. hal-00655478

HAL Id: hal-00655478

https://inria.hal.science/hal-00655478v1

Submitted on 29 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Tradeoff exploration between reliability, power consumption, and execution time

Ismail Assayad[1], Alain Girault[2], and Hamoudi Kalla[3]

[1] ENSEM (RTSE team), University Hassan II of Casablanca, Morroco.
[2] INRIA and Grenoble University (POP ART team and LIG lab), France.
[3] University of Batna (SECOS team), Algeria.

Abstract. We propose an off-line scheduling heuristics 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 heuristics, 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.

1 Introduction

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 present an off-line scheduling heuristics that, from a given software application graph and a given multiprocessor architecture, 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). We target homogeneous distributed architecture, such as multicore processors. Our tricriteria scheduling heuristics 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. However, DVFS 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. The two criteria *length* and *reliability* are thus *antagonistic* with each other and with the schedule length, which makes this problem all the more difficult.

Let us address the issues raised by multicriteria optimization. Figure 1 illustrates the particular case of two criteria to be minimized. Each point x^1 to x^7 represents a solution, that is, a different tradeoff between the Z_1 and Z_2 criteria: the points x^1, x^2, x^3, x^4, and x^5 are *Pareto optima* [17]; the points x^2, x^3, and x^4 are *strong optima* while the points x^1 and x^5 are *weak op-*

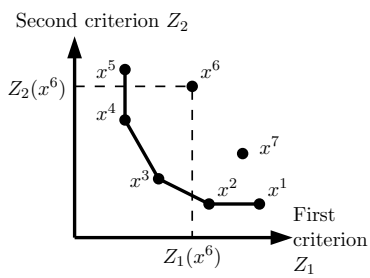

Fig. 1: Pareto front for a bicriteria minimization problem.

tima. The set of all Pareto optima is called the *Pareto front*.

It is fundamental to understand that no single solution among the points x^2, x^3, and x^4 (the strong Pareto optima) can be said, a priori, 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. This is why we advocate producing, for a given problem instance, the Pareto front rather than a single solution. Since we have three criteria, it will be a *surface* in the 3D space (length,reliability,power).

The main contribution of this paper is TSH, the *first* tricriteria scheduling heuristics able to produce a Pareto front in the space (length,reliability,power), 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 (length,reliability) heuristics called BSH [6]. The tricriteria extension presented in this paper is necessary because of the crucial impact of the voltage on the failure probability.

2 Principle of the method and overview

To produce a Pareto front, the usual method involves transforming all the criteria except one into *constraints*, and then minimizing the last remaining criterion iteratively [17]. 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. Another way is to slice the interval $[0, +\infty)$ into a finite number of contiguous sub-intervals of the form $[K_1^i, K_1^{i+1}]$.

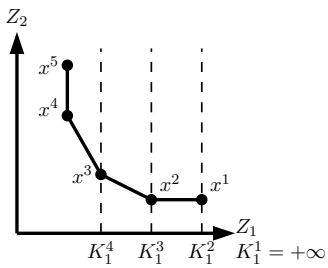

Fig. 2: Transformation method to produce the Pareto front.

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 have to use *list scheduling heuristics*, which have demonstrated their good performances in the past [10]. We propose in this paper a suitable list scheduling heuristics, adapted from [6].

Using list scheduling to minimize a criterion Z_2 under the constraint that another criterion Z_1 remains below some threshold K_1 (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: 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. As a consequence, if we attempt to use the energy as a constraint (i.e., $Z_1{=}E$) and the schedule length as a criteria to be minimized (i.e., $Z_2{=}L$), then we are bound to fail. Indeed, the fact that all the scheduling decisions made at the stage of any intermediary schedule

S' meet the constraint $E(S')<K_1$ *cannot* guarantee that the final schedule S will meet the constraint $E(S)<K_1$. 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: it is neither an increasing nor a decreasing function of the schedule. So the same reasoning applies if the reliability is taken as a constraint. This is why we take instead, as a criterion, the *global system failure rate per time unit* (GSFR), first defined in [6]. By construction, the GSFR is an invariant measure of the schedule's reliability (see Section 3.7).

For these reasons, each run of our tricriteria scheduling heuristics TSH minimizes the schedule length under the double constraint that the power consumption and the GSFR remain below some thresholds, respectively P_{obj} and Λ_{obj}. By running TSH with decreasing values of P_{obj} and Λ_{obj}, starting with $+\infty$ and $+\infty$, 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 heuristics minimizing the schedule length under the constraints of both the reliability and the power.

3 Models

3.1 Application algorithm graph

Most embedded real-time systems are reactive, and therefore consist of some algorithm executed periodically, triggered by a periodic execution clock. Our model is therefore that of an application algorithm graph $\mathcal{A}lg$ which is repeated infinitely. $\mathcal{A}lg$ is an *acyclic oriented graph* $(\mathcal{X}, \mathcal{D})$ (See Figure 3(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. Operations with no predecessor (resp. successor) are called *input* operations (resp. *output*). Operations do not have any side effect, except for input/output operations: an input operation (resp. output) is a call to a sensor driver (resp. actuator).

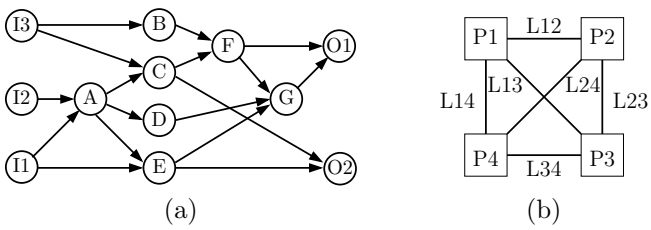

Fig. 3. (a) An example of algorithm graph $\mathcal{A}lg$: I_1, I_2, and I_3 are input operations, O_1 and O_2 are output operations, A-G are regular operations; (b) An example of an architecture graph $\mathcal{A}rc$ with four processors, P_1 to P_4, and six communication links.

The $\mathcal{A}lg$ graph is acyclic but it is infinitely repeated in order to take into account the reactivity of the modeled system, that is, its reaction to external stimuli produced by its environment.

3.2 Architecture model

We assume that the architecture is an *homogeneous and fully connected multiprocessor* 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 3(b)). $\mathcal{P}$ 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. 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*.

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 $\mathcal{E}xe : (\mathcal{X} \times \mathcal{P}) \cup (\mathcal{D} \times \mathcal{L}) \mapsto \mathbb{R}^+$ giving the *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).

The WCET analysis is the topic of much work [18]. 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 [16].

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 $\mathcal{A}rc$. It consists of two functions: the *spatial allocation function* Ω gives, for each operation of $\mathcal{A}lg$ (resp. for each data-dependency), the subset of processors of $\mathcal{A}rc$ (resp. the subset of communication links) that will execute it; and the *temporal allocation function* Θ gives the starting date of each operation (resp. each data-dependency) on its processor (resp. its communication link): $\Omega : \mathcal{X} \mapsto 2^{\mathcal{P}}$ and $\Theta : \mathcal{X} \times \mathcal{P} \mapsto \mathbb{R}^+$.

In this work we only deal with *static* schedules, for which the function Θ is 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 for each data-dependency), $|\Omega(X)|=1$. In contrast, a schedule is *with (active) replication* if for some operation X (or some data-dependency), $|\Omega(X)|\geq 2$. The number $|\Omega(X)|$ is called the *replication factor* of X. A schedule is

partial if not all the operations 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 $\mathcal{A}rc$. For a schedule S, we note it $L(S)$.

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}. For each operation, its WCET assumes that the processor operates at f_{max} and V_{max} (and similarly for the WCCT of the data-dependencies). Because the circuit delay is almost linearly related to $1/V$ [3], 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, 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 (taken as a scaling factor) is:

$$\mathcal{E}xe(X, C, f) = \mathcal{E}xe(X, C)/f \tag{1}$$

The power consumption P of a single operation placed on a single processor is computed according to the classical model of Zhu et al. [19]:

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

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 [5]), 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, which is a simplifying assumption, since one would normally need to take into account the actual switching activity of each operation to compute accurately the consummed energy. However, such an accurate computation is infeasible for the application sizes we consider here.

For a multiprocessor schedule S, we cannot apply directly Eq (2). Instead, we must compute the total energy $E(S)$ consumed by S, and then divide by the schedule length $L(S)$:

$$P(S) = E(S)/L(S) \tag{3}$$

We compute $E(S)$ by summing the contribution of each processor, depending on the voltage and frequency of each operation placed onto it. On the processor p_i, the energy consumed by each operation is the product of the active power P^i_{ind}+P^i_d by its execution time. As a conclusion, the total consumed energy is:

$$E(S) = \sum_{i=1}^{|\mathcal{P}|} \left(\sum_{o_j \in p_i} (P^i_{ind} + P^i_d).\mathcal{E}xe(o_j, p_i) \right) \tag{4}$$

3.6 Failure hypothesis

Both processors and communication links can fail, and they are *fail-silent* (a behavior which can be achieved at a reasonable cost [1]). Classically, we adopt the failure model of Shatz and Wang [15]: failures are *transient* and the maximal duration of a failure is such that it affects only the current operation executing onto the faulty processor; 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 $10^{-6}/hr$ [1].

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 [19]. 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 [9].

The *reliability* of a system is defined as the probability that it operates correctly during a given time interval. According to our model, the reliability of the processor P (resp. the communication link L) during the duration d is $R{=}e^{-\lambda d}$. 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:

$$R(X,C) = e^{-\lambda_C \, \mathcal{E}xe(X,C)} \tag{5}$$

From now on, the function R will either be used with two variables as in Eq (5), 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).

3.7 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 [6]. The GSFR of a static schedule S, noted $\Lambda(S)$, is computed by Eq (6):

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} \text{ with } R(S) = \prod_{(o_i,p_j)\in S} R(o_i,p_j) \text{ and } U(S) = \sum_{(o_i,p_j)\in S} \mathcal{E}xe(o_i,p_j) \tag{6}$$

Eq (6) 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 of S (by definition of the reliability, Section 3.6). Eq (6) also uses the total processor utilization $U(S)$ instead of the schedule length $L(S)$, so that the GSFR can be computed *compositionally*. According to Eq (6), 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 1 in [6]).

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 that the GSFR criterion is usable in practice.

4 The tricriteria scheduling algorithm TSH

4.1 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 [12] (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 [19]. 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:

$$\lambda(f) = \lambda_0.10^{\frac{b(1-f)}{1-f_{min}}} \tag{7}$$

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

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 to give the supply voltage of the processor for each operation: $\Omega : \mathcal{X} \mapsto \mathcal{Q}$, where $\mathcal{Q}$ is the domain of the sets of pairs $\langle p, v\rangle \in \mathcal{P} \times \mathcal{V}$.

Figure 4 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}. Since we do not apply DVFS to the communication links, we only compute the energy consumed by the processors (see Eq (4)):

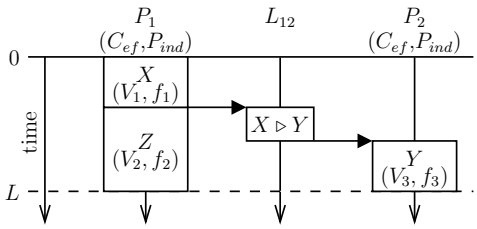

Fig. 4: A simple schedule of length L.

- On P_1: $E(P_1) = P_{ind}L + C_{ef}V_1^2 f_1\, \mathcal{E}xe(X, P_1, f_1) + C_{ef}V_2^2 f_2\, \mathcal{E}xe(Z, P_1, f_2)$.
- On P_2: $E(P_2) = P_{ind}L + C_{ef}V_3^2 f_3\, \mathcal{E}xe(Y, P_1, f_3)$.

By applying Eqs (1) and (3), we thus obtain:

$$P(S)=\frac{E(P_1)+E(P_2)}{L}=2P_{ind}+\frac{C_{ef}}{L}.\left(V_1^2\mathcal{E}xe(X,P_1)+V_2^2\mathcal{E}xe(Z,P_1)+V_3^2\mathcal{E}xe(Y,P_2)\right)$$

The general formula for a schedule S is therefore:

$$P(S)=|\mathcal{P}|.P_{ind}+\frac{C_{ef}}{L(S)}\sum_{i=1}^{|\mathcal{P}|}\left(\sum_{o_j\in p_i}V(o_j)^2.\mathcal{E}xe(o_j,p_i)\right) \tag{8}$$

4.2 Decreasing the GSFR

According to Eq (6), *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). 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).

To compute the GSFR of a static schedule with replication, we use *Reliability Block-Diagrams (RBD)* [11]. 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 or a data-dependency $X \triangleright Y$ placed onto a communication link L. The reliability of a block is therefore computed according to Eq (5).

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 step of the scheduling heuristics, 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 (6).

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* [6]. The idea is that, for each data dependency $X \triangleright Y$ such that is 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 5).

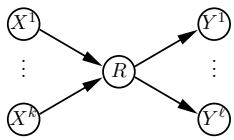

Fig. 5: A routing operation.

4.3 Scheduling heuristics

To obtain the Pareto front in the space (length,GSFR,power), we pre-define 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 scheduling heuristics TSH presented below.

TSH is a *greedy 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.

TSH works with two lists of operations of $\mathcal{A}lg$: the candidate operations $\mathcal{O}^{(n)}_{cand}$ and the already scheduled operations $\mathcal{O}^{(n)}_{sched}$. The superscript (n) denotes the current iteration of the scheduling algorithm. One operation is scheduled at each iteration. Initially, $\mathcal{O}^{(0)}_{sched}$ is empty while $\mathcal{O}^{(0)}_{cand}$ contains the input operations of $\mathcal{A}lg$. At any iteration (n), all the operations in $\mathcal{O}^{(n)}_{cand}$ are such that all their predecessors are in $\mathcal{O}^{(n)}_{sched}$.

The power-efficient schedule pressure is a variant of the schedule pressure cost function [8], 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 operation o_i, and each processor p_j as:

$$\sigma^{(n)}(o_i, p_j) = ETS^{(n)}(o_i, p_j) + LTE^{(n)}(o_i) - CPL^{(n-1)} \tag{9}$$

where $CPL^{(n-1)}$ is the critical path length of the partial schedule 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.

First, we generalize the schedule pressure σ to a set of processors:

$$\sigma^{(n)}(o_i, \mathcal{P}_k) = ETS^{(n)}(o_i, \mathcal{P}_k) + LTE^{(n)}(o_i) - CPL^{(n-1)} \tag{10}$$

where $ETS^{(n)}(o_i, \mathcal{P}_k)$=$\max_{p_j \in \mathcal{P}_k} ETS^{(n)}(o_i, p_j)$.

Then, we consider the schedule length as a criterion to be minimized, and the GSFR and the power as two constraints to be met: for each candidate operation $o_i \in \mathcal{O}^{(n)}_{cand}$, we compute the best subset of pairs $\langle \text{processor}, \text{voltage} \rangle$ to execute

o_i with the *power-efficient schedule pressure* of Eq (11): $\mathcal{Q}^{(n)}_{best}(o_i) = \mathcal{Q}_j$ s.t.:

$$\sigma^{(n)}(o_i, \mathcal{Q}_j) = \min_{\mathcal{Q}_k \in \mathcal{Q}} \left\{ \sigma^{(n)}(o_i, \mathcal{Q}_k) | \Lambda^{(n)}(o_i, \mathcal{Q}_k) \leq \Lambda_{obj} \wedge P^{(n)}(o_i, \mathcal{Q}_k) \leq P_{obj} \right\} \quad (11)$$

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 4.1), and $\Lambda^{(n)}(o_i, \mathcal{Q}_k)$ (resp. $P^{(n)}(o_i, \mathcal{Q}_k)$) is the GSFR (resp. the power consumption) of the partial schedule after replicating and scheduling o_i on all the processors of $\mathcal{Q}_k$ with their respective specified voltages. When computing $\Lambda^{(n)}(o_i, \mathcal{Q}_k)$, the failure rate of each processor is computed by Eq (7) according to its voltage in $\mathcal{Q}_k$. Finally, $P^{(n)}(o_i, \mathcal{Q}_k)$ is computed by Eq (8).

To guarantee that the constraint $\Lambda^{(n)}(o_i, \mathcal{Q}_k) \leq \Lambda_{obj}$ is met, the subset $\mathcal{Q}_k$ is selected such that the GSFR of the parallel macro-block that contains the replicas of o_i on the processors of $\mathcal{Q}_k$ is less than Λ_{obj}. If this last macro-block B is such that $\Lambda(B) \leq \Lambda_{obj}$ and if $\Lambda^{(n-1)} \leq \Lambda_{obj}$, then $\Lambda^{(n)} \leq \Lambda_{obj}$ (thanks to the invariance property of the GSFR).

Similarly, the subset $\mathcal{Q}_k$ is selected such that the power constraint $P^{(n)}(o_i, \mathcal{Q}_k) \leq P_{obj}$ is met. There can exist several valid possibilities for the subset $\mathcal{Q}_k$ (valid in the sense that the power constraint is met). However, some of them may lead to the impossibility of finding a valid schedule for the *next* scheduled operation, during step $n+1$. In particular, this is the case when the next scheduled operation *does not* increase the schedule length, because it fits in a slack of the previous schedule: $L^{(n+1)} = L^{(n)}$. At the same time, the total energy increases strictly because of the newly scheduled operation: $E^{(n+1)} > E^{(n)}$. By hypothesis, we have $P^{(n)} = E^{(n)}/L^{(n)} \leq P_{obj}$, but it follows that $P^{(n+1)} = E^{(n+1)}/L^{(n+1)} = E^{(n+1)}/L^{(n)} > E^{(n)}/L^{(n)} = P^{(n)}$, so even though $P^{(n)} \leq P_{obj}$, it may very well be the case that $P^{(n+1)} > P_{obj}$. To prevent this and 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 P_{max}. P_{max} is the computed power under the highest frequency f_m such that $P_{ind} + P_{max} = P_{ind} + C_{ef} V^2 f \leq P_{obj}/N$, where N is the processors number. If the consumed power with f_m exceeds P_{obj}, then the next highest operating frequency $f \leq f_m$ is selected, and so on. 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 6. For lack of space, we do not study in this paper the impact of this over-estimation on the total schedule length.

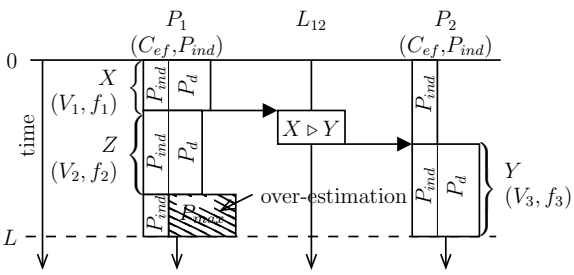

Fig. 6: Over-estimation of the energy consumption.

Once we have computed, for each candidate operation o_i of $\mathcal{O}^{(n)}_{cand}$, the best subset of pairs $\langle \text{processor}, \text{voltage} \rangle$ to execute o_i, with the power-efficient sched-

ule pressure of Eq (11), we compute the *most urgent* of these operations by:

$$o_{urg} = o_i \in \mathcal{O}_{cand}^{(n)} \text{ s.t. } \sigma^{(n)}\big(o_i, \mathcal{Q}_{best}^{(n)}(o_i)\big) = \max_{o_j \in \mathcal{O}_{cand}^{(n)}} \left\{ \sigma^{(n)}\big(o_j, \mathcal{Q}_{best}^{(n)}(o_j)\big) \right\} \tag{12}$$

Finally, we schedule this most urgent operation o_{urg} on the processors of $\mathcal{Q}_{best}^{(n)}(o_j)$, and we finish the current iteration (n) by updating the lists of scheduled and candidate operations: $\mathcal{O}_{sched}^{(n)} := \mathcal{O}_{sched}^{(n-1)} \cup \{o_{urg}\}$ and $\mathcal{O}_{cand}^{(n+1)} := \mathcal{O}_{cand}^{(n)} - \{o_{urg}\} \cup \{t' \in succ(o_{urg}) \mid pred(t') \subseteq \mathcal{O}_{sched}^{(n)}\}$.

5 Simulation results

We perform two kinds of simulations. Firstly, Figure 7 shows 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 [6]. 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 $\lambda_\ell = 5.10^{-4}$; these values are reasonnable for modern fail-silent processors [1]; the set of supply voltages is $\mathcal{V} = \{0.25, 0.50, 0.75, 1.0\}$ (scaling factor).

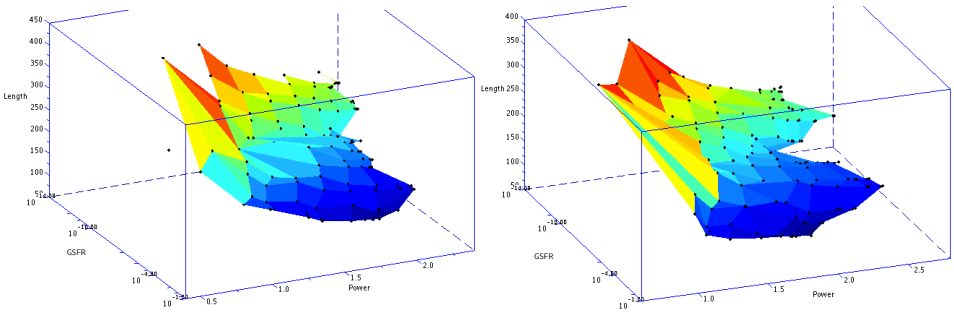

Fig. 7. Pareto front generated for a random graph of 30 operations on 3 processors (left) or 4 processors (right).

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 decreasing values of P_{obj} and Λ_{obj}, starting with $+\infty$ and $+\infty$, are selected from two sets of values: $\Lambda_{obj} \in \{\alpha.10^{-\beta}\}$ where $\alpha \in \{4, 8\}$ and $\beta \in \{1, 2, \ldots 20\}$, and $P_{obj} \in \{0.8, 0.6, 0.4, 0.2\}$. TSH being a heuristics, changing the parameters of this grid could change locally some points of the Pareto front, but not its overall shape.

The two figures 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.

Figure 7 exposes to the designer a choice of several tradeoffs between the execution time, the power consumption, and the reliability level. For instance in Figure 7 (right), 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, we can obtain schedules with a shorter execution length even though we impose identical constraints to the GSFR and the power.

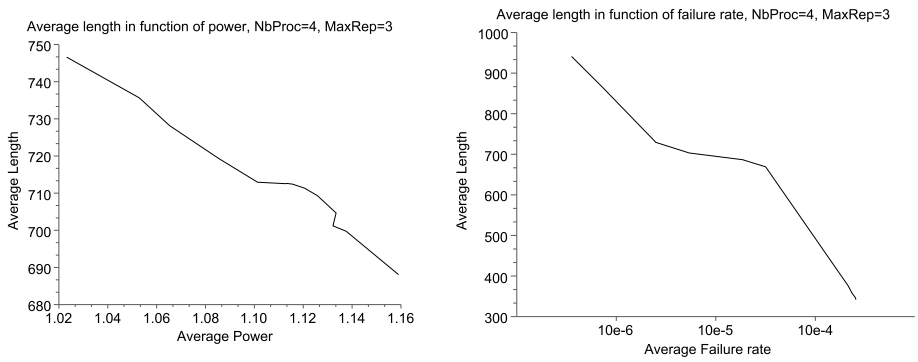

Fig. 8. Average schedule length in function of the power (left) or the GSFR (right).

Secondly, Figure 8 shows how the schedule length varies, respectively in function of the required power consumption (left) or of the required GSFR (right). Both curves are 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. Similarly, the average schedule length increases when the constraint Λ_{obj} on the GSFR decreases. Again, the two criteria, schedule length and GSFR, are antagonistic.

6 Related work

Many solutions exist in the literature to optimize the schedule length and the energy consumption (e.g., [13]), or to optimize the schedule length and the reliability (e.g., [4, 7, 2]), but very few tackle the problem of optimizing the *three* criteria (length,reliability,energy). The closest to our work are [19, 14].

Zhu et al. have studied the impact of the supply voltage on the failure rate [19], 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 sys-

tem 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 [14]. Both length and reliability are taken as a constraint. 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.

7 Conclusion

We have presented a new off-line tricriteria scheduling heuristics, called TSH, to minimize the schedule 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. Both the power and the GSFR are taken as *constraints*, so 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, 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. TSH is an extension of our previous bicriteria (length,reliability) heuristics BSH [6]. The tricriteria extension is necessary because of the crucial impact of the voltage on the failure probability.

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 the exponential.

References

1. 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.
2. 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*, San Diego (CA), USA, September 2010.
3. 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.

4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. J.Y-T. Leung, editor. *Handbook of Scheduling. Algorithms: Models, and Performance Analysis.* Chapman & Hall/CRC Press, 2004.
11. D. Lloyd and M. Lipow. *Reliability: Management, Methods, and Mathematics*, chapter 9. Prentice-Hall, 1962.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms.* Springer-Verlag, 2006.
18. 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.
19. 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.