HAL
archives-ouvertes.fr

A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate

Alain Girault, Hamoudi Kalla

▶ To cite this version:

Alain Girault, Hamoudi Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Transactions on Dependable and Secure Computing, Institute of Electrical and Electronics Engineers, 2009, 6 (4), pp.241–254. <hal-00746768>

HAL Id: hal-00746768
https://hal.inria.fr/hal-00746768

Submitted on 29 Oct 2012

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.

A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate

Alain Girault and Hamoudi Kalla

Abstract—We propose a new framework for the (length,reliability) bicriteria static multiprocessor scheduling problem. Our first criterion remains the schedule's length, crucial to assess the system's real-time property. For our second criterion, we consider the global system failure rate, seen as if the whole system were a single task scheduled onto a single processor, instead of the usual reliability, because it does not depend on the schedule length like the reliability does (due to its computation in the classical exponential distribution model). Therefore, we control better the replication factor of each individual task of the dependency task graph given as a specification, with respect to the desired failure rate.

To solve this bicriteria optimization problem, we take the failure rate as a constraint, and we minimize the schedule length. We are thus able to produce, for a given dependency task graph and multiprocessor architecture, a Pareto curve of non-dominated solutions, among which the user can choose the compromise that fits his requirements best.

Compared to the other bicriteria (length,reliability) scheduling algorithms found in the literature, the algorithm we present here is the first able to improve significantly the reliability, by several orders of magnitude, making it suitable to safety critical systems.

Index Terms—Reliability, bicriteria optimization, Pareto optima, static multiprocessor scheduling, reliability block-diagrams, safety-critical systems.

1 Introduction

Bicriteria (length,reliability) multiprocessor scheduling has recently attracted a lot of attention [1], [7], [8], [13], [21], [22]. The purpose is to schedule statically a graph of tasks (also called operations) onto a distributed memory multiprocessor architecture, such that two criteria of the obtained schedule are optimized: its *length* (crucial to assess the system's real-time property) and its *reliability* (crucial to assess the system's dependability). With the ever growing research field on critical embedded systems, this domain remains very important.

We use the widely accepted reliability model of Shatz and Wang [23], where each hardware component (processor or communication link) is fail-silent and is characterized by a *constant failure rate per time unit* λ, such that its reliability during the interval of time d be $e^{-\lambda d}$ (that is, the occurrences of the failures follow a constant parameter Poisson law; this is also known as the exponential distribution model [3]). The chosen mean to increase the reliability of a system is 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). Intuitively, adding more replicas increases the reliability, but also in general the schedule length: in this sense, we say that the two criteria are *antagonistic*.

But things are not so easy! We show that using together the reliability criterion and the schedule length criterion raises technical difficulties, because the reliability depends intrinsically on the duration of the operations and communications. For instance, choosing a processor such that the duration d of a given operation is smaller (which is good for the length criterion) induces a higher reliability (which is also good for the reliability criterion); this is because the $d \mapsto e^{-\lambda d}$ function is decreasing. It follows that it is difficult to design a satisfactory bicriteria scheduling heuristic. In particular, this has three drawbacks: first, the length criterion overpowers the reliability criterion; second, it is very tricky to control precisely the replication factor of the operations onto the processors, from the beginning to the end of the schedule (in particular, it can cause a "funnel" effect); and third, the reliability is not a monotonous function of the schedule.[1] Yet, this is the approach that has been followed so far in the literature.

For this reason, we propose a new criterion in place of the schedule reliability, which we call the *global system failure rate (GSFR)*. 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. Since the failure rate is "per time unit", it is intrinsically independent of the duration of the operations. As we will show, our new theoretical framework is consistent with the intuition that replication is good for reliability but bad for length. This is our first contribution: Section 4

- *Alain Girault is with INRIA and Grenoble University (POP ART project-team and LIG laboratory), France, Email: Alain.Girault@inria.fr.* *This research was supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.*

- *Hamoudi Kalla is with the University of Batna (SECOS team), Algeria, Email: Hamoudi.Kalla@univ-batna.dz.*

Manuscript received October 10, 2007; revised June 27, 2008.

1. If S' is a prefix schedule of S, then the reliability of S is not necessarily greater than the reliability of S'.

motivates the GSFR and explains how to compute it.

Using the GSFR is also very satisfactory in the area of periodically executed schedules. This is the case in most real-time embedded systems, which are periodically sampled systems. In such cases, applying brutally the exponential reliability model yields very low reliabilities due to very long execution times (the same remark applies also to very long schedules). Hence one has to compute beforehand the desired reliability of a single iteration from the global reliability of the system during its full mission; but this computation depends on the total duration of the mission (which is known) and on the duration of one single iteration (which may not be known because it depends on the length of the schedule under construction). In contrast, the GSFR remains *constant* during the whole system's mission: the GSFR during a single iteration is by construction identical to the GSFR during the whole mission.

Now, let us address the issues raised by bicriteria optimization. In Figure 1, each point x^1 to x^7 represents a solution of a bicriteria minimization problem: the points x^1, x^2, x^3, x^4, and x^5 are *Pareto optima* [27]; the points x^1 and x^5 are *weak optima* while the points x^2, x^3, and x^4 are *strong optima*. The set of all Pareto optima is called the *Pareto curve*. Then, several approaches exist to tackle bicriteria optimization problems (these methods extend naturally to multicriteria) [27]:

1. Aggregation of the two criteria into a single one, so as to transform the problem into a classical single criterion optimization one.

2. 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.

3. 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.

4. Interaction with the user, in order to guide the search for a Pareto optimum.

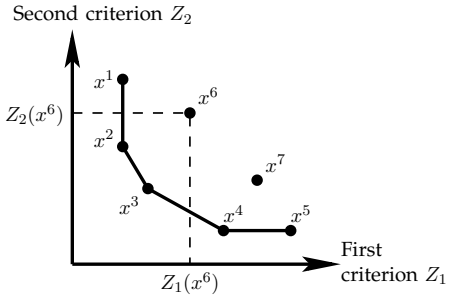

Fig. 1. Pareto optima and Pareto curve for a bicriteria minimization problem.

Our second contribution belongs to the second case: we propose a new static scheduling algorithm that takes as a *constraint* a given GSFR, and that attempts to *minimize* the schedule length on an distributed heterogeneous architecture, by using a greedy list scheduling with a smart cost function. By executing several times this algorithm on a given instance of the problem (that is, a given algorithm and a given architecture), each time with a different GSFR, we obtain a set of points (length,GSFR). From this set of points, we can extract the subset of non-dominated points. By definition, this subset is the approximated Pareto curve of the given instance of the problem. We present in details this new algorithm (Section 5) and show extensive simulation results (Section 6).

2 Related work

Numerous works have dealt specifically with the bicriteria (length,reliability) static scheduling problem for distributed memory heterogeneous architectures. The RDLS algorithm [7] ("Reliable Dynamic Level Scheduling") is an extension of the Dynamic Level Scheduling (DLS) algorithm [24]. The latter is a list scheduling heuristic that takes as input a DAG of operations $\mathcal{A}lg$ and an heterogeneous set of processors $\mathcal{A}rc$. At each step, the heuristic evaluates all the pairs $\langle o, p\rangle$ and chooses to place the operation o on processor p such that the cost function $DL(o,p)$ be maximal. An additional term is added to the DL cost function so as to take into account the reliability of the schedule (under the classical reliability model of Shatz and Wang). Hence, the length and reliability criteria are combined into a single criterion. The simulation results show that RDLS is always better than DLS w.r.t. the reliability criterion, but always worse w.r.t. the length criterion. Note that operations nor data-dependencies are never replicated.

[8] proposes a bicriteria extension of the DLS algorithm by building, at each scheduling step, two lists containing all the pairs $\langle v_i, m_j\rangle$ of tasks v_i ready to be scheduled and of machines m_j: the first list is ordered by decreasing order of the DL function to take into account the increase in the length of the schedule resulting from the decision to place v_i on m_j; the second list is ordered by increasing order of the $\Delta COST$ function to take into account the decrease in reliability resulting from this same decision. Therefore, each pair $\langle v_i, m_j\rangle$ has a rank in each of those lists, respectively noted $Rank^1_{i,j}$ and $Rank^2_{i,j}$. These two ranks are then combined as $Rank_{i,j} = \delta^1 Rank^1_{i,j} + \delta^2 Rank^2_{i,j}$, where the two numbers δ^1 and δ^2 are chosen empirically so as to balance the two criteria. Finally, the pair $\langle v_i, m_j\rangle$ having the smallest $Rank_{i,j}$ is selected and v_i is placed on m_j. Here again, they do not replicate operations nor data-dependencies.

[13] proposes a Biobjective Scheduling Algorithm (BSA) similar to RDLS: the two criteria are aggregated into a single biobjective cost function, the reliability model is the one of Shatz and Wang, and the tasks are not replicated to increase the reliability. $f(t,p)$ and $\sigma(t,p)$ denote respectively the finish time and the reliability of the task t when it is scheduled on the processor p. The biobjective function is $D(t,p) = \sqrt{\theta\left(\frac{f(t,p)}{max_p f(t,p)}\right)^2 + (1-\theta)\left(\frac{\sigma(t,p)}{max_p \sigma(t,p)}\right)^2}$. The normalization with the max prevents one criterion from dominating the other one, like in [1]. Compared to RDLS, the reliability is better but the schedule length is worse.

The eFCRD algorithm ("Efficient Fault-Tolerant Reliability Cost-Driven Algorithm") proposed [22] produces static fault-tolerant schedules by replicating each operation twice (one primary replica and one secondary replica that sometimes overlaps with other operations); hence no more than one processor failure can be tolerated. Concerning the reliability (Shatz and Wang model), eFCRD tries to schedule each primary replica on a processor such that the increase of the reliability cost be minimal and that the deadline of the task be met. However, the reliability of the communication links is not taken into account.

[21] addresses the tricriteria optimization problem (length,reliability,energy) on an heterogeneous architecture with a reliable bus. Both the length and the reliability are taken as a constraint. The energy consumption is minimized thanks to dynamic voltage scaling, but since this increases the execution time, it also has an impact on the reliability due to the $e^{-\lambda d}$ reliability formula. As a result, the criteria are intrinsically dependent and the problems mentioned in the introduction arise. Also, it is assumed that the user will specify the number of processor failures to be tolerated (with re-execution or passive replication) in order to satisfy the desired reliability constraint.

[9] tackles the problem of maximizing the reliability and minimizing the makespan on related machines where processors are subject to crash fault. The tasks are not replicated, hence the increase in reliability is very limited. For independent tasks with unitary execution time, the authors propose an optimal scheduling algorithm and compute the approximate Pareto, which they further generalize for non-unitary independent tasks. Finally, for the general case, they propose the RHEFT algorithm which improves over the well-known HEFT scheduling algorithm (Heterogeneous Earliest Finish Time).

In a previous paper [1], we have proposed a bicriteria scheduling heuristic, where each candidate operation o is tested onto all possibles subsets of processors $\mathcal{P}$. For each choice $\langle o, \mathcal{P} \rangle$, we compute the induced variation in length $\Delta\ell$ and reliability Δr, both normalized w.r.t. the length upper bound and the reliability lower bound constraints (both provided by the user). In the bicriteria plane of Figure 1, we project each point $\langle \Delta\ell, \Delta r \rangle$ onto the diagonal line of angle θ, and choose the point whose projection is closest to $\langle 0, 0 \rangle$ (this amounts to aggregating the two criteria). By varying θ, we can give more weight to the length or the reliability criterion. Note that when the subset $\mathcal{P}$ is a singleton, the operation o is not replicated, while when it is a set $\{p_1, \ldots, p_k\}$, the operation o is actively replicated onto the processors p_1 to p_k.

Among the works presented so far, only [1], [21], [22] replicate the tasks to increase the reliability. In contrast, [7]–[9], [13] cannot increase the reliability of the obtained system above the reliability level of the hardware platform the system is executed on, because the operations are not replicated. In contrast, we use the active replication of tasks and data-dependencies to achieve any reliability level required by the user, at the price of some schedule length overhead of course.

Also, all the works presented so far suffer from the three drawbacks mentioned in the introduction: first, the length criterion overpowers the reliability criterion; second, it is very tricky to control precisely the replication factor of the operations onto the processors, from the beginning to the end of the schedule; and third, the reliability is not a monotonous function of the schedule. Our new framework solves these problems thanks to the replacement of the reliability criterion by a new criterion that does not depend on the execution times of the operations and data-dependencies.

Finally, numerous works have restricted themselves to acyclic networks [15], [16], [18], [23], because the time necessary to compute the probability that a path is operational becomes linear in the length of the path (this is known as the *terminal pair problem* [4], which is NP-hard in the case of a general network). Moreover, all these works explore the state space entirely in order to find the optimal allocation of the modules onto the processors, for the reliability criterion. Even with space reduction techniques [18], this exhaustive exploration remains very costly, hence they are only capable of treating very small size problems (less than 8 modules).

3 Preliminaries

3.1 Application algorithm graph

An application algorithm graph $\mathcal{A}lg$ is an *acyclic oriented graph* $(\mathcal{X}, \mathcal{D})$. Its nodes (the set $\mathcal{X}$) are software blocks called *operations*. They do not have any side effect, except for input/output operations: an input operation is a call to a sensor driver, while an output operation is a call to an actuator driver. Each arc of $\mathcal{A}lg$ (the set $\mathcal{D}$) is a *data-dependency* between two operations. If X▷Y is a data-dependency, then X is a *predecessor* operation of operation Y, while Y is a *successor* operation of operation X. Operation X is also called the *source* of the data-dependency X▷Y, and Y is its *destination*. The set of all predecessors of X is noted $pred(\mathrm{X})$. The set of all successors of X is noted $succ(\mathrm{X})$.

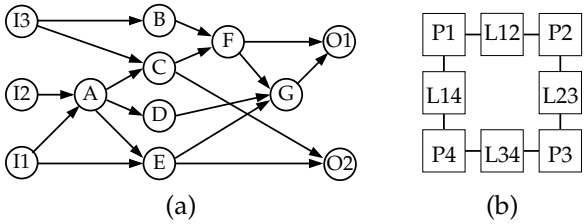

Fig. 2. (a) An example of algorithm graph $\mathcal{A}lg$: I1, I2, and I3 are input operations, O1 and O2 are output operations, A-G are regular operations; (b) An example of an architecture graph $\mathcal{A}rc$ with four processors, P1 to P4, and four communication links, L12 to L34.

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. Each execution of $\mathcal{A}lg$ is called an *iteration*. The operations with no predecessor are called *input* operations: they are sensor driver invocations to read data from the environment. The operations with no successor are called *output* operations: they are actuator driver invocations to emit data towards the environment.

3.2 Architecture graph

The architecture graph $\mathcal{A}rc$ 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 $\mathcal{A}$. $\mathcal{P}$ 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 $\mathcal{A}rc$ always connects one processor and one communication link.

We say that the graph $\mathcal{A}rc$ is *connected* if there exists a path between any two processors, and *complete* if there exists a communication link between any two processors. The architecture is *completely connected* if its graph $\mathcal{A}rc$ is complete. The bipartite graph representation is essential to model in a uniform way point-to-point communication links and multi-point communication media and buses. In our work, the only assumption we make on the network topology is that it must be connected.

3.3 Execution characteristics

Along with the algorithm graph $\mathcal{A}lg$ and the architecture graph $\mathcal{A}rc$, we are also given a table $\mathcal{E}xe$ of the *worst-case execution times* (WCET) of all the operations of $\mathcal{A}lg$ onto all the processors of $\mathcal{A}rc$, and the *worst-case communication times* (WCCT) of all the data-dependencies of $\mathcal{A}lg$ onto all the communication links of $\mathcal{A}rc$. An intra-processor communication takes no time to execute. 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 [5] or with caches and pipelines [26]. In particular, it has been applied to the most critical embedded system, namely the AIRBUS A380 avionics software running on the MOTOROLA MPC755 processor [10], [25].

Finally, the architecture is *homogeneous* if all its processors and all its communication links have the same characteristics (speed, memory, reliability, bandwidth...). Otherwise it is *heterogeneous*. Our proposed method and bicriteria scheduling algorithm BSH works with heterogeneous architectures.

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 $\mathcal{A}rc$. Its 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). In this work we only deal with *static* schedules, for which the function Θ is static, and our schedules are computed *off-line*.

A static schedule is *without replication* if for each operation X (and for each data-dependency), $|\Omega(\mathrm{X})| = 1$. In contrast, a schedule is *with (active) replication* if for some operation X (or some data-dependency), $|\Omega(\mathrm{X})| \geq 2$. The number $|\Omega(\mathrm{X})|$ is called the *replication factor* of X.

In a schedule with replication, a given operation can either start its execution as soon as it has received all its input data from *at least one* replica of each of its predecessors, or it can await until it has received all its input data from *all* the replicas of each of its predecessors. The first case is the best choice when one wants to minimize the schedule length; this choice has even been used by several task duplication algorithms to further minimize the schedule length by adding additional replicas for the sole purpose of avoiding costly inter-processor communications (e.g., [6]). However, the second case is the best choice when one wants to maximize the reliability, since the exact reliability of the schedule can be computed in polynomial time, instead of in exponential time in the first case (this has to do with the form of the reliability block-diagrams; see Section 3.6 below).

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* or *makespan* of a schedule is the $\max$ of the termination times of the last operation scheduled on each of the processors of $\mathcal{A}rc$. We note it $\mathtt{C_{max}}$.

3.5 Failure hypothesis

Both processors and communication links can fail, and they are *fail-silent*. Classically, we adopt the failure model of Shatz and Wang [23]: 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. Besides, 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 $10^{-6}/hr$.

Moreover, failure occurrences are statistically independent events. Note that transient failures 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 [28].

The *reliability* of a system measures its continuity of service. It is defined as the probability that it functions correctly during a given time interval [2]. 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:

$$R(X, C) = e^{-\lambda_C \, \mathcal{E}xe(X,C)} \quad (1)$$

In the sequel, the function R will either be used with two variables as in Equation (1), or with only one variable to denote the reliability of a schedule (or a part of a schedule).

3.6 Computing the reliability of a schedule

Definitions: *Reliability Block-Diagrams* (RBD) have been introduced to compute the reliability of a system [3], [20]. Formally, an RBD is an *acyclic oriented graph* (N, E), for which 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* iff 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 therefore equal to 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▷Y placed onto a communication link L. The reliability of a block is therefore computed according to Equation (1).

Computing the reliability in this way assumes that the occurrences of the failures are *statistically independent events*. Without this hypothesis, the fact that some blocks belong to several paths from S to D makes the computation of the reliability very complex. Concerning hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [19].

The main drawback of this approach is that the computation of the reliability is, in general, exponential in the size of the RBD. When the schedule is without replication, the RBD is *serial* (i.e., there is a single path from S to D) so the computation of the reliability is linear in the size of the RBD. But when the schedule is with replications, the RBD has no particular form, so the computation of the reliability is exponential in the size of the RBD.

An RBD is *parallel* if all its blocks are in parallel, and is *serial-parallel* if its consists of a sequence of parallel portions. In both cases, the computation of the reliability is also linear in the size of the RBD.

As an example, consider the simple $\mathcal{A}lg$ graph of Figure 3. A possible schedule with replication of this $\mathcal{A}lg$ graph is show in Figure 5(a), where operation X is replicated twice (its replicas being noted X^1 and X^2) and operation Y is also replicated twice (its replicas being noted Y^1 and Y^2). The RBD corresponding to this schedule is shown in Figure 4. It has no particular form.

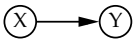

Fig. 3. A simple $\mathcal{A}lg$ graph X→Y.

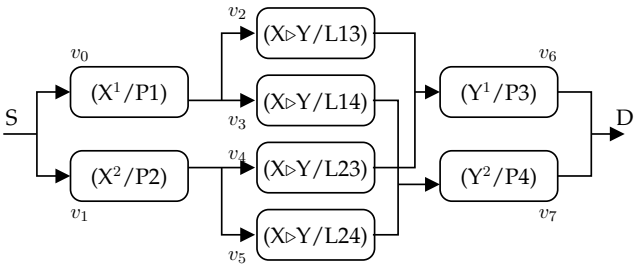

Fig. 4. The RBD of the schedule of Figure 5(a).

Minimal cut sets method: One solution to compute efficiently the reliability of a system from its RBD involves computing all its *minimal cut sets* [17]. A *cut set* in an RBD is a set of blocks $\mathcal{C}$ such that there exist no path from S to D if we remove all the blocks of $\mathcal{C}$ from the RBD. A cut $\mathcal{C}$ is *minimal* if, whatever the block that is removed from it, the resulting set is not a cut anymore. The reliability of the static multiprocessor schedule S is approximated by the reliability of the RBD composed of all the minimal cuts put in sequence. The reliability of a minimal cut set $\mathcal{C}_i$ is the reliability of all its blocks put in parallel: $R(\mathcal{C}_i) = 1 - \prod_{(o,c)\in\mathcal{C}_i}(1 - R(o, c))$. Hence, in order to approximate the reliability $R(S)$ of an RBD, it suffices to compute all its minimal cut sets $\mathcal{C}_i$, numbered from 1 to n, and then to compute the reliability $R^-(S)$ of the serial-parallel RBD composed of all these cuts. This computation is linear in the number of minimal cut sets:

$$R(S) \geq R^-(S) = \prod_{i=1}^{n}\Big(1 - \prod_{(o,c)\in\mathcal{C}_i}\Big(1 - R(o, c)\Big)\Big) \quad (2)$$

This approximation is a lower bound, so if a schedule S is such that $R^-(S) > R_{obj}$, where R_{obj} is the reliability objective that the target schedule must meet, then it proves that $R(S)$ is greater than R_{obj}, which is perfectly fine. The main problem of this method is that, depending on the structure of the RBD, the total number of minimal cuts is between $n - 1$ and 2^{n-2}, where n is the number of nodes of the RBD. For instance, the RBD of Figure 4 has 11 minimal cut sets: $\{v_0, v_1\}$, $\{v_2, v_3, v_4, v_5\}$, $\{v_6, v_7\}$, $\{v_0, v_4, v_5\}$, $\{v_1, v_2, v_3\}$, $\{v_2, v_4, v_7\}$, $\{v_3, v_5, v_6\}$, $\{v_0, v_5, v_6\}$, $\{v_0, v_4, v_7\}$, $\{v_1, v_3, v_6\}$, and $\{v_1, v_2, v_7\}$. Hence, the approximate computation of the reliability with the minimal cut sets method is also exponential in the size of S. This is a drawback of the bicriteria scheduling algorithm of [1].

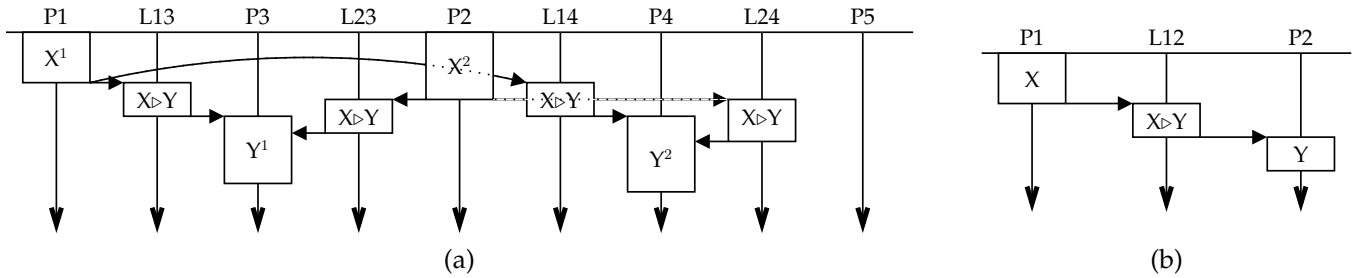

Fig. 5. (a) A simple schedule without routing of the $\mathcal{A}lg$ graph of Figure 3; (b) A simple schedule without replication of the $\mathcal{A}lg$ graph of Figure 3.

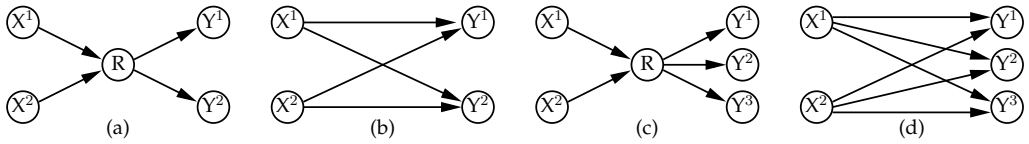

Fig. 6. (a) A transformed $\mathcal{A}lg$ graph with replication and with a routing operation R when X and Y are replicated twice; (b) Same as (a) without the routing operation; (c) Same as (a) when Y is replicated thrice; (d) Same as (c) without the routing operation.

BDD-based methods: To make this computation faster, several tools use BDDs (e.g., SHARPE [14] or ALTARICA [11]). For instance, the BDD coding the RBD of Figure 4, as produced by SHARPE, is shown in Figure 7. It has 17 leaves, which means that the exact reliability formula is a sum of 17 terms (much less than the theoretical maximum number of 256 leaves/terms corresponding to its 8 variables, v_0 to v_7).

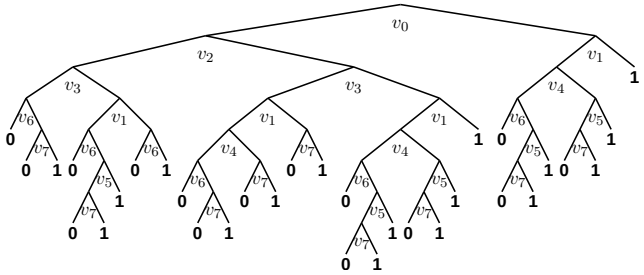

Fig. 7. BDD coding the RBD of Figure 4.

Serial-parallel schedule method: Our scheduling algorithm BSH (see Section 5) must compute the reliability of partial schedules a great number of times at each iteration, hence this reliability computation must be extremely fast. For this reason, we propose to produce schedules in such a way that the corresponding RBD will be, by construction, serial-parallel, and furthermore with a number of blocks exactly equal to the number of nodes in the schedule. This allows us to compute its reliability in a linear time w.r.t. the size of S. For instance, the serial-parallel RBD of the same example as the BDD of Figure 7 has only 9 nodes, so computing the reliability is even faster than with the BDD-based method. The drawback is that the schedules we produce have a greater $\mathtt{C}_{\mathtt{max}}$, that is, a higher length overhead. Indeed, unlike what we have said in Section 3.4 regarding the schedules with replication, our proposal forces each operation to wait for the reception of *all* its input data sent by all the replicas of its predecessors before starting its execution (otherwise the obtained RBD would not be serial-parallel). However, we will see in Section 6.7 that the additional overhead incurred by this design choice is reasonable.

Concerning the scheduling algorithm, this method involves, for any data-dependency X▷Y, inserting an additional *routing operation*, whose WCET is equal to 0 time unit, between the data-dependencies coming out of the source operation X and the data-dependencies going towards the destination operation Y. Figure 6(a) shows the graph transformation that occurs from the $\mathcal{A}lg$ graph of Figure 3 when X is replicated twice (X^1 and X^2) and Y is replicated twice (Y^1 and Y^2): a routing operation R is inserted between the two replicas of X and the two replicas of Y. This guarantees that each replica of Y will receive its input data from both replicas of X before starting its execution. The graph of Figure 6(a) has four communications. In comparison, when no routing operation is inserted, Figure 6(b) shows the transformed $\mathcal{A}lg$ graph: it also has four communications, but they are *concurrent*, while in Figure 6(a), the routing operation limits the concurrency: indeed, the two communications received by R must be scheduled before the two communications sent by R. This difference in concurrency explains the additional length overhead incurred by the routing operations. Figures 6(c) and 6(d) show a similar situation where X is replicated twice and Y is replicated thrice. In this case, the $\mathcal{A}lg$ graph with the routing operation has five communications, while the $\mathcal{A}lg$ graph without routing operation has six communications. Again, there is less concurrency between the communications in Figure 6(c) than 6(d), but in this case there are also fewer communications. This explains why the additional length overhead is limited (see Section 6.7).

4 Computing the global system failure rate per time unit (GSFR)

4.1 Definition of the GSFR

Using the reliability as a scheduling criterion raises technical difficulties because the reliability depends intrinsically on the duration of the operations and communications (Equation (1)). More precisely, the function $d \mapsto e^{-\lambda d}$ being decreasing, a bicriteria (length,reliability) cost function will tend to give a greater importance to the length criterion than to the reliability criterion. It follows that it is difficult to design a satisfactory bicriteria scheduling heuristic. In particular, it is very tricky to control precisely the replication factor of the operations from the beginning to the end of the schedule, and to avoid funnel effects.

For this reason, we propose a new criterion in place of the schedule reliability, namely the *failure rate per time unit of the global system*, defined as follows:

Definition 4.1 (GSFR) *Let $\mathcal{A}lg$ be an algorithm graph, $\mathcal{A}rc$ be an architecture graph, and S be a static multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$. The global system failure rate (GSFR) of S, noted $\Lambda(S)$, is the failure rate of S seen as if it were a single operation scheduled onto a single processor. Let U be the total utilization of the hardware resources by S, and R be its reliability, then:*

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} \quad with \quad U(S) = \sum_{(o_i,p)\in S} \mathcal{E}xe(o_i,p) \quad (3)$$

Since the failure rate is "per time unit", it is intrinsically independent of the duration of the operations. As a consequence, our new theoretical framework is consistent with the intuition that replication is good for the reliability but bad for the length. Also, the definition of $U(S)$ is consistent with the "hot" failure model. 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.

4.2 A schedule without replication

Consider again the very simple $\mathcal{A}lg$ graph of Figure 3, scheduled onto an $\mathcal{A}rc$ graph composed of two processors P1 and P2, connected by a point-to-point communication link L12. Figure 5(b) shows one possible schedule without replication of this $\mathcal{A}lg$ graph, where each operation (resp. communication) is represented by a box whose height is proportional to its execution time (resp. communication time) onto its processor (resp. its communication link).

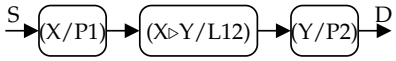

Fig. 8. The RBD of the schedule of Figure 5(b).

Let λ_1 and λ_2 be the failure rates of P1 and P2, and λ_{12} be the failure rate of L12. Let t^1_{X}=$\mathcal{E}xe(\mathrm{X},\mathrm{P1})$, t^2_{Y}=$\mathcal{E}xe(\mathrm{Y},\mathrm{P2})$, and t^{12}_{XY}=$\mathcal{E}xe(\mathrm{X} \triangleright \mathrm{Y},\mathrm{L12})$. By applying Equation (1), we obtain $R(\mathrm{X},\mathrm{P1})$=$e^{-\lambda_1 t^1_{\mathrm{X}}}$, $R(\mathrm{X} \triangleright \mathrm{Y},\mathrm{L12})$=$e^{-\lambda_{12} t^{12}_{\mathrm{XY}}}$, and $R(\mathrm{Y},\mathrm{P2})$=$e^{-\lambda_2 t^2_{\mathrm{Y}}}$.

Figure 8 shows the RBD corresponding to the schedule of Figure 5(b). Since its three blocks are in sequence in the RBD, we have the following global reliability:

$$\begin{aligned} R(S) &= R(\mathrm{X},\mathrm{P1}) \cdot R(\mathrm{X} \triangleright \mathrm{Y},\mathrm{L12}) \cdot R(\mathrm{Y},\mathrm{P2}) \\ &= e^{-\lambda_1 t^1_{\mathrm{X}}} \cdot e^{-\lambda_{12} t^{12}_{\mathrm{XY}}} \cdot e^{-\lambda_2 t^2_{\mathrm{Y}}} \\ &= e^{-(\lambda_1 t^1_{\mathrm{X}} + \lambda_{12} t^{12}_{\mathrm{XY}} + \lambda_2 t^2_{\mathrm{Y}})} \end{aligned}$$

By applying Equation (3), the GSFR of this system S is:

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} = \frac{\lambda_1 t^1_{\mathrm{X}} + \lambda_{12} t^{12}_{\mathrm{XY}} + \lambda_2 t^2_{\mathrm{Y}}}{t^1_{\mathrm{X}} + t^{12}_{\mathrm{XY}} + t^2_{\mathrm{Y}}} \quad (4)$$

In the particular case where the architecture is homogeneous w.r.t. the reliability, that is, if λ_1=λ_2=λ_{12}, then $\Lambda(S)$=λ_1. This result is perfectly consistent with our definition of the GSFR.

4.3 A schedule with replication

We now consider a schedule with replication of the same $\mathcal{A}lg$ graph of Figure 3, scheduled onto an $\mathcal{A}rc$ graph composed of five processors P1 to P5, completely connected by point-to-point communication links. Figure 9 shows one possible schedule with replication of $\mathcal{A}lg$ onto $\mathcal{A}rc$, where two active replicas of both X and Y (respectively noted X^1, X^2, Y^1, and Y^2) are scheduled. Note that the execution time of the routing operation R is 0, so it is represented by a box whose height is 0. In this schedule, all the operations are placed onto distinct processors, but of course this is not necessarily the case. Actually, BSH makes its best to group some operations on the same processor so as to avoid inter-processor communications. The RBD of this schedule is shown in Figure 10.

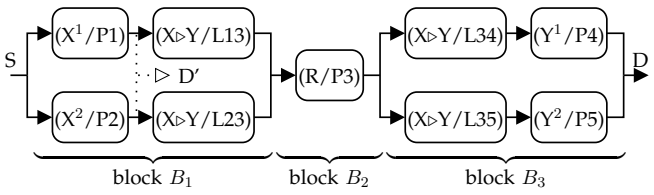

Fig. 10. The RBD of the schedule of Figure 9.

First, we consider the partial schedule S' composed only of the first two operations X^1 and X^2 in the schedule of Figure 9. Its RBD is also shown in Figure 10, but with destination D′. Since those two blocks are in parallel, the reliability of S' is (we adopt the same compact notations as in Section 4.2, e.g., t^2_{X}=$\mathcal{E}xe(\mathrm{X},\mathrm{P2})$):

$$\begin{aligned} R(S') &= 1 - \big(1 - R(\mathrm{X}^1,\mathrm{P1})\big) \cdot \big(1 - R(\mathrm{X}^2,\mathrm{P2})\big) \\ &= 1 - \left(1 - e^{-\lambda_1 t^1_{\mathrm{X}}}\right) \cdot \left(1 - e^{-\lambda_2 t^2_{\mathrm{X}}}\right) \end{aligned} \quad (5)$$

For the sake of the example, we take the following numbers: λ_1=λ_2=10^{-5} and t^1_{X}=t^2_{X}=5. We thus obtain:

$$\left.\begin{array}{l} R(\mathrm{X}^1,\mathrm{P1}) = 0.9999500 \\ R(\mathrm{X}^2,\mathrm{P2}) = 0.9999500 \end{array}\right\} \Rightarrow R(S') = 0.999999997500$$

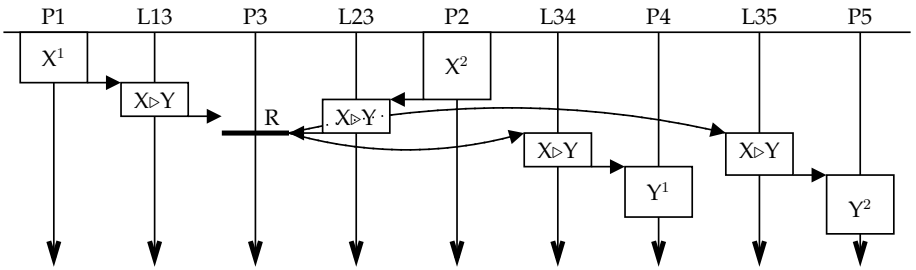

Fig. 9. A simple schedule with replication of the $\mathcal{A}lg$ graph of Figure 3.

To compute the GSFR $\Lambda(S')$, we apply Equation (3):

$$\Lambda(S') = \frac{-\log R(S')}{U(S')} \tag{6}$$

In our case, $U(S') = 10$ time units $(5+5)$. Hence:

$$\Lambda(S') = 2.49987\,10^{-10}$$

We can notice that, when $x \simeq 0$, $e^x \simeq 1+x$, or equivalently $1-e^{-x} \simeq x$, hence Equation (5) becomes:

$$R(S') \simeq 1 - (\lambda_1 t^1_{\mathrm{X}}) \cdot (\lambda_2 t^2_{\mathrm{X}}) \tag{7}$$

By a similar reasoning, when $x \simeq 0$, $\log(1-x) \simeq -x$, hence Equation (6) becomes:

$$\Lambda(S') \simeq \frac{\lambda_1 t^1_{\mathrm{X}} \lambda_2 t^2_{\mathrm{X}}}{U(S')} = \frac{\lambda_1 t^1_{\mathrm{X}} \lambda_2 t^2_{\mathrm{X}}}{t^1_{\mathrm{X}} + t^2_{\mathrm{X}}} \tag{8}$$

This approximation can be easily verified in the above computation. Equation (8) can be generalized to n active replicas of X scheduled in parallel onto n processors P1,...,Pn, with failure rates $\lambda_1, \ldots, \lambda_n$, and such that the $\mathcal{E}xe(\mathrm{X}^i, \mathrm{P}i)$=$t^i_{\mathrm{X}}$. This yields:

$$\Lambda(S') \simeq \frac{\prod_{i=1}^{n} \lambda_i t^i_{\mathrm{X}}}{\sum_{i=1}^{n} t^i_{\mathrm{X}}} \tag{9}$$

The crucial point is that the *order of magnitude* of the GSFR is the product of the failure rates of the n processors, divided by the sum of the execution times. Hence, adding active replicas to a schedule greatly increases it GSFR, which is consistent with the intuition.

We now consider the full schedule S. The RBD is a sequence of three blocks, B_1, B_2, and B_3:

$$R(B_1) = 1 - \left(1 - e^{-(\lambda_1 t^1_{\mathrm{X}} + \lambda_{13} t^{13}_{\mathrm{XY}})}\right) \cdot \left(1 - e^{-(\lambda_2 t^2_{\mathrm{X}} + \lambda_{23} t^{23}_{\mathrm{XY}})}\right)$$

$$R(B_2) = 1 \text{ because the WCET of R is always 0}$$

$$R(B_3) = 1 - \left(1 - e^{-(\lambda_{34} t^{34}_{\mathrm{XY}} + \lambda_4 t^4_{\mathrm{X}})}\right) \cdot \left(1 - e^{-(\lambda_{35} t^{35}_{\mathrm{XY}} + \lambda_5 t^5_{\mathrm{X}})}\right)$$

The schedule's reliability is $R(S) = R(B_1) \cdot R(B_2) \cdot R(B_3)$. For the sake of the example, we take the following numbers: $\forall i, \lambda_i$=10^{-5}, $\forall i,j, \lambda_{ij}$=$10^{-4}$, $\forall i, t^i_{\mathrm{X}}$=$t^i_{\mathrm{Y}}$=5, and $\forall i,j, t^{ij}_{\mathrm{XY}}$=3. We thus obtain:

$$\left.\begin{array}{r} R(B_1) = 0.99999988 \\ R(B_2) = 1 \\ R(B_3) = 0.99999988 \end{array}\right\} \Longrightarrow R(S) = 0.99999976$$

According to the WCET of the individual operations and data-dependencies, the utilization of the schedule S is equal to 32 time units, which yields:

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} = 7.500\,10^{-9}$$

As expected from the graph transformation (illustrated in Figure 6), the active replication of X and Y decreases the GSFR, but the addition of the routing operation increases the $\mathtt{C}_{\mathtt{max}}$ (it would have been 13 time units instead of 16). Of course, when computing the multiprocessor schedule, BSH tries to group some operations on the same processor so as to avoid inter-processor communications (by putting Y^1 onto P1 for instance).

4.4 Mixed serial-parallel schedules

First, let us recall the two important formulas obtained in Sections 4.2 and 4.3. Both formulas concern a RBD composed of two macro-blocks B_1 and B_2, with respective GSFR Λ_1 and Λ_2, and respective utilization T_1 and T_2:

$$\text{serial schedule:} \quad \Lambda(B_1 \cdot B_2) = \frac{\Lambda_1 T_1 + \Lambda_2 T_2}{T_1 + T_2} \tag{10}$$

$$\text{parallel schedule:} \quad \Lambda(B_1 \| B_2) \simeq \frac{\Lambda_1 T_1 \Lambda_2 T_2}{T_1 + T_2} \tag{11}$$

Like in the previous subsections, we consider a schedule of the same $\mathcal{A}lg$ graph of Figure 3, where X is replicated twice and Y is not replicated. This schedule is actually a subset of the one shown in Figure 9, where processor P5 and link L35 are removed, and where the replica Y^1 is replaced by the non-replicated operation Y. The RBD starts with two blocks in parallel (block B_{1_1} for X^1/P1 and B_{1_2} for X^2/P2, and we note B_1=$B_{1_1} \| B_{1_2}$) followed by a sequence of three blocks (block B_{2_1}=R, B_{2_2}=X▷Y/L34, and B_{2_3}=Y/P4). Hence, by applying Equations (10) and (11), we obtain:

$$\Lambda_{1_1} = \frac{\lambda_1 t^1_{\mathrm{X}} + \lambda_{13} t^{13}_{\mathrm{XY}}}{t^1_{\mathrm{X}} + t^{13}_{\mathrm{XY}}} \qquad T_{1_1} = t^1_{\mathrm{X}} + t^{13}_{\mathrm{XY}}$$

$$\Lambda_{1_2} = \frac{\lambda_2 t^2_{\mathrm{X}} + \lambda_{23} t^{23}_{\mathrm{XY}}}{t^2_{\mathrm{X}} + t^{23}_{\mathrm{XY}}} \qquad T_{1_2} = t^2_{\mathrm{X}} + t^{23}_{\mathrm{XY}}$$

$$\Lambda_1 \simeq \frac{\Lambda_{1_1} T_{1_1} \Lambda_{1_2} T_{1_2}}{T_{1_1} + T_{1_2}} \qquad T_1 = T_{1_1} + T_{1_2}$$

$$\Lambda_2 = \frac{0 + \lambda_{34} t_{XY}^{34} + \lambda_4 t_Y^4}{0 + t_{XY}^{34} + t_Y^4} \qquad T_2 = 0 + t_{XY}^{34} + t_Y^4$$

$$\Lambda = \frac{\Lambda_1 T_1 + \Lambda_2 T_2}{T_1 + T_2} \qquad T = T_1 + T_2$$

For the sake of the example, let us take the following numbers: $\forall i, \lambda_i{=}10^{-5}$, $\forall i,j, \lambda_{ij}{=}10^{-4}$, $\forall i, t_X^i{=}t_Y^i{=}5$, and $\forall i,j, t_{XY}^{ij}{=}3$. This yields $T_{1_1}{=}T_{1_2}{=}T_2{=}8$, $T_1{=}16$, $T{=}24$, $\Lambda_{1_1}{=}\Lambda_{1_2}{=}\Lambda_2$, $\Lambda_1 \simeq \frac{1}{2} T_{1_1} \Lambda_{1_1} \Lambda_{1_2}$, and $\Lambda{=}\frac{1}{3}(2\Lambda_1 + \Lambda_2)$. The numerical computations give $\Lambda_{1_1}{=}\Lambda_{1_2}{=}\Lambda_2{=}4.375\,10^{-5}$, $\Lambda_1 \simeq 7.656\,10^{-9}$, and $\Lambda \simeq 2.917\,10^{-5}$.

The order of magnitude of Λ_1 is 10^{-9} while that of Λ_2 is 10^{-5}. This is expected since Λ_1 is the GSFR of two blocks in parallel, while Λ_2 is the GSFR of two blocks in sequence (we do not count R whose GSFR is equal to 1). The order of magnitude of their sum is 10^{-5}, which means that Λ_2 "wins" over Λ_1, but the $\frac{1}{3}$ multiplicative factor allows us to "regain" partially an order of magnitude.

It is interesting to generalize this RBD to n macro-blocks in sequence, where the first macro-block corresponds to a single non-replicated operation, while the remaining $n-1$ macro-blocks are each composed of two blocks in parallel (the corresponding operation being replicated twice). Assuming, for the sake of simplicity, that the execution times of all the blocks are all equal to 5, the GSFR is then $\Lambda{=}\frac{1}{n}(\sum_{i=1}^{n} \Lambda_i)$. Let us also assume that the failure rates of all the processors are all equal to 10^{-4}; we then have $\Lambda_1{=}10^{-4}$ (no replication in the first macro-block) and $\forall\, 2 \leq i \leq n, \Lambda_i{=}2.5\,10^{-8}$ (one replication in the $n-1$ other macro-blocks). Depending on n, we thus have the GSFR shown in Figure 11.

It follows that, in our example, if one operation is not replicated, then six other operations must be replicated twice if we want to "regain" one order of magnitude.

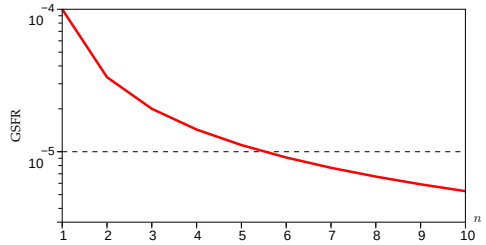

Fig. 11. GSFR of a system composed of n blocks in sequence.

5 BICRITERIA SCHEDULING ALGORITHM

5.1 Principle

Our bicriteria scheduling algorithm is a *greedy list scheduling heuristic* called BSH. It takes as input an algorithm graph $\mathcal{A}lg$, a heterogeneous architecture graph $\mathcal{A}rc$, the table $\mathcal{E}xe$ of all operations WCET and communications WCCT, and a constraint Λ_{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}, and such that its $\mathtt{C_{max}}$ is as small as possible. BSH uses the *active replication of operations* to meet the Λ_{obj} constraint, and the *dependable schedule pressure* as a cost function to minimize the $\mathtt{C_{max}}$. Besides, it inserts *routing operations* to make sure that the RBD of any partial schedule is serial-parallel. According to these principles, BSH is based on the following proposition:

Proposition 5.1 *In a serial-parallel RBD, if each macro-block in the sequence is such that its GSFR is less than Λ_{obj}, then the GSFR of the whole RBD is also less than Λ_{obj}.*

Proof: Let $(B_i)_{1 \leq i \leq n}$ be the n parallel macro-blocks that are in sequence in the RBD. By hypothesis, we have:

$$\begin{aligned}
& \forall 1 \leq i \leq n,\ \Lambda(B_i) \leq \Lambda_{obj} \\
\Rightarrow\ & \forall 1 \leq i \leq n,\ \Lambda(B_i)U(B_i) \leq \Lambda_{obj} U(B_i) \\
\Rightarrow\ & \sum_{i=1}^{n} \Lambda(B_i)U(B_i) \leq \sum_{i=1}^{n} \Lambda_{obj} U(B_i) \\
\Rightarrow\ & \sum_{i=1}^{n} \Lambda(B_i)U(B_i) \leq \Lambda_{obj} \times \sum_{i=1}^{n} U(B_i) \qquad (12)
\end{aligned}$$

Since the macro-blocks B_i are in sequence, we compute the GSFR for the whole RBD thanks to Equation (10):

$$\Lambda = \frac{\sum_{i=1}^{n} \Lambda(B_i)U(B_i)}{\sum_{i=1}^{n} U(B_i)}$$

Hence, Inequality (12) implies $\Lambda \leq \Lambda_{obj}$. □

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

The dependable schedule pressure is a variant of the schedule pressure cost function σ proposed in [12], 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 follows:

$$\sigma^{(n)}(o_i, p_j) = ETS^{(n)}(o_i, p_j) + LTE^{(n)}(o_i) - CPL^{(n-1)} \qquad (13)$$

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. When computing the length of this path, since the operations are not scheduled yet, we do not know their actual WCET, so we compute the average WCET of each operation on all processors.

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

$$\sigma^{(n)}(o_i, \mathcal{P}_k) = ETS^{(n)}(o_i, \mathcal{P}_k) + LTE^{(n)}(o_i) - CPL^{(n-1)} \qquad (14)$$

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 makespan as a criterion to be minimized and the GSFR as a constraint to be met: for each candidate operation $o_i \in L^{(n)}_{cand}$, we compute the best subset of processors to execute o_i with the dependable schedule pressure of Equation (15):

$$\mathcal{P}^{(n)}_{best}(o_i) = \mathcal{P}_j \in 2^{\mathcal{P}} \text{ s.t.} \quad (15)$$
$$\sigma^{(n)}(o_i, \mathcal{P}_j) = \min_{\mathcal{P}_k \in 2^{\mathcal{P}}} \left\{ \sigma^{(n)}(o_i, \mathcal{P}_k) \,|\, \Lambda^{(n)}(o_i, \mathcal{P}_k) \leq \Lambda_{obj} \right\}$$

where $\Lambda^{(n)}(o_i, \mathcal{P}_k)$ is the GSFR of the partial schedule after replicating and scheduling o_i on all the processors of $\mathcal{P}_k$, and $2^{\mathcal{P}}$ is the set of all subsets of $\mathcal{P}$. To guarantee the constraint $\Lambda^{(n)}(o_i, \mathcal{P}_k) \leq \Lambda_{obj}$, the subset $\mathcal{P}_k$ is selected such that the GSFR of the parallel block that contains the replicas of o_i on the processors of $\mathcal{P}_k$ is less than Λ_{obj} (see Figure 10). If this last block B is such that $\Lambda(B) \leq \Lambda_{obj}$ and if $\Lambda^{(n-1)} \leq \Lambda_{obj}$, then thanks to Proposition 5.1, $\Lambda^{(n)} \leq \Lambda_{obj}$.

Finally, we compute the most urgent operation with Equation (16):

$$o_{urg} = o_i \in L^{(n)}_{cand} \text{ s.t.} \quad (16)$$
$$\sigma^{(n)}\left(o_i, \mathcal{P}^{(n)}_{best}(o_i)\right) = \max_{o_j \in L^{(n)}_{cand}} \left\{ \sigma^{(n)}\left(o_j, \mathcal{P}^{(n)}_{best}(o_j)\right) \right\}$$

Computing the GSFR of the partial schedule has been explained in Section 4. However, when doing so, we must take into account the *future* communications for the data-dependencies sent by the last scheduled operation Y. Indeed, according to Figure 6, these communications will be replicated into the same number of replicas as Y, but the links where they will be placed will only be known at the next iteration of BSH. As a consequence, for an $\mathcal{A}lg$ graph of the form X→Y→Z, we build the RBD of Figure 12, where the future communications are in red/dotted.

Furthermore, when selecting the links to compute the reliability of the blocks corresponding to these future communications (L′ and L″ in Figure 12), we choose the links which have the worse failure rates, in order to guarantee that, whatever the scheduling choice made during the next iteration $n+1$ of BSH, $\Lambda^{(n+1)}$ will be less than Λ_{obj}.

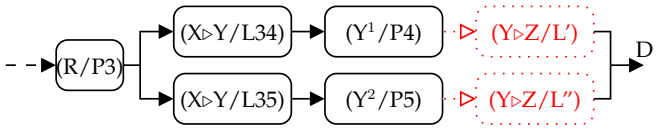

Fig. 12. A RBD taking into account the future communications of Y.

5.2 Bicriteria scheduling heuristic (BSH)

The BSH scheduling heuristic is shown in Figure 13. Initially, $L^{(0)}_{sched}$ is empty and $L^{(0)}_{cand}$ is the list of operations without any predecessors. At the n-th iteration, these lists are updated according to the data-dependencies of $\mathcal{A}lg$.

At each iteration n, one operation o_i of the list $L^{(n)}_{cand}$ is selected to be scheduled. For this, we select at the micro-steps ① and ②, for each candidate operation o_i, the best subset of processors $\mathcal{P}^{(n)}_{best}(o_i)$ to replicate and schedule o_i, such that the GSFR of the resulting partial schedule is less than Λ_{obj}. Then, among those best pairs $\langle o_i, \mathcal{P}^{(n)}_{best}(o_i)\rangle$, we select at the micro-step ③ the one having the biggest dependable schedule pressure value, i.e., the most urgent pair $\langle o_{urg}, \mathcal{P}^{(n)}_{best}(o_{urg})\rangle$.

Algorithm BSH:
input: $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{E}xe$, and Λ_{obj};
output: a multi-processor static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$ that minimizes the makespan and satisfies Λ_{obj}, or a failure message;
begin
Compute the set $2^{\mathcal{P}}$ of all subsets of $\mathcal{P}$;
/* the user can limit the degree k of processor combinations */

$L^{(0)}_{cand}$:= {operations without predecessors};
$L^{(0)}_{sched} := \emptyset$;
$n := 0$;
while $L^{(n)}_{cand} \neq \emptyset$ **do**

① For each candidate operation $o_i \in L^{(n)}_{cand}$, compute $\sigma^{(n)}(o_i, \mathcal{P}_k)$ for each subset $\mathcal{P}_k$ of $2^{\mathcal{P}}$.

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

$$\sigma^{(n)}(o_i, \mathcal{P}^{(n)}_{best}(o_i)) = \min_{\mathcal{P}_k \in 2^{\mathcal{P}}} \left\{ \sigma^{(n)}(o_i, \mathcal{P}_k) \,| \right.$$
$$\left. \Lambda^{(n)}(o_i, \mathcal{P}_k) \leq \Lambda_{obj} \right\}$$

③ Select the most urgent candidate operation o_{urg} among all o_i of $L^{(n)}_{cand}$ such that:

$$\sigma^{(n)}\left(o_{urg}, \mathcal{P}^{(n)}_{best}(o_{urg})\right) = \max_{o_j \in L^{(n)}_{cand}} \left\{ \sigma^{(n)}\left(o_j, \mathcal{P}^{(n)}_{best}(o_j)\right) \right\}$$

④ Schedule each replica of o_{urg} on each processor of $\mathcal{P}^{(n)}_{best}(o_{urg})$;

⑤ **if**$(\mathcal{P}^{(n)}(o_{urg}) = \emptyset)$ **then**
return"fails to satisfy failure rate objective"; **exit**;
/* the user can modify Λ_{obj} or k and re-run BSH */

⑥ Update the lists of candidate and scheduled operations:
$L^{(n)}_{sched} := L^{(n-1)}_{sched} \cup \{o_{urg}\}$;
$L^{(n+1)}_{cand} := L^{(n)}_{cand} - \{o_{urg}\} \cup$
$\{t' \in succ(o_{urg}) \,|\, pred(t') \subseteq L^{(n)}_{sched}\}$;

⑦ $n := n + 1$;

end while
end

Fig. 13. The BSH scheduling heuristic.

The selected operation o_{urg} is replicated and scheduled at the micro-step ④ on each processor of $\mathcal{P}_{best}(o_{urg})$ computed at micro-step ②, and the communications implied by these assignments are also replicated and scheduled according to the graph transformations of Figures 6(a) and 6(c).

We check at the micro-step ⑤ if the failure rate objective is satisfied or not. If it is not, the user can lower the failure rate objective Λ_{obj} or increase the maximal degree of processor combinations k, and re-execute BSH.

Finally, we update the lists of candidate and scheduled operations at the micro-step ⑥.

Let n be the number of operations in the graph $\mathcal{A}lg$ and p be the number of processors in the graph $\mathcal{A}rc$. Since BSH is a greedy list scheduling algorithm, the number of steps necessary to complete is exactly equal to n. The only computation step that is not in $\mathcal{O}(1)$ is step ①, which takes $\mathcal{O}(2^p)$. As a result, the overall time complexity of BSH is:

$$\mathcal{T}(BSH) = \mathcal{O}\left(n \times 2^p\right) \tag{17}$$

However, by limiting the degree of processor combinations to k, step ① becomes $\mathcal{O}(\sum_{i=1}^{k} C_p^i)$ (where C_p^i is the binomial coefficient), so the overall time complexity of BSH becomes:

$$\mathcal{T}(BSH) = \mathcal{O}\left(n \times \sum_{i=1}^{k} C_p^i\right) \tag{18}$$

To obtain an approximate Pareto curve, one just needs to run BSH several times. With the computing power available in modern computers, the computation time is reasonable (but not for simulations made on large numbers of graphs).

6 Simulation Results

6.1 Target architecture

We have conducted extensive simulations of our BSH algorithm. The following figures have been obtained by generating randomly 50 $\mathcal{A}lg$ graphs of 50 operations each, with a Communication-to-Computation Ratio set to 1.[2] Each of these $\mathcal{A}lg$ graphs were then given to BSH with two heterogeneous and completely connected $\mathcal{A}rc$ graphs, having 4 and 6 processors (resp. named P1 to P4 and P1 to P6). Table 1 gives the individual failure rates per time unit of all the processors and communication links in the two $\mathcal{A}rc$ graphs.

P1,P2	P3,P4	L12,L13,L14,L23,L24,L34
$\lambda_{1,2} = 10^{-4}$	$\lambda_{3,4} = 10^{-5}$	$\lambda_m = 10^{-3}$

P5,P6	L15,L16,L25,L26,L35,L36,L45,L46,L56
$\lambda_{5,6} = 5.10^{-5}$	$\lambda_m = 10^{-3}$

TABLE 1
Failure rates per time unit in the two $\mathcal{A}rc$ graphs.

6.2 Variation of the GSFR w.r.t. Λ_{obj}

Figure 14 shows the actual GSFR obtained by BSH w.r.t. the objective Λ_{obj}, averaged over the 50 $\mathcal{A}lg$ graphs. The successive values of Λ_{obj} given to BSH were: 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, and 10^{-7}.

When Λ_{obj} is within $[10^{-7}, 10^{-4}]$, $\Lambda(S)$ is very close to Λ_{obj}. When Λ_{obj} is within $[10^{-3}, 10^{-2}]$, $\Lambda(S)$ is significantly lower than Λ_{obj}. Indeed, it is not possible to obtain such a bad level of failure rate because the processors and communication links of $\mathcal{A}rc$ are too reliable.

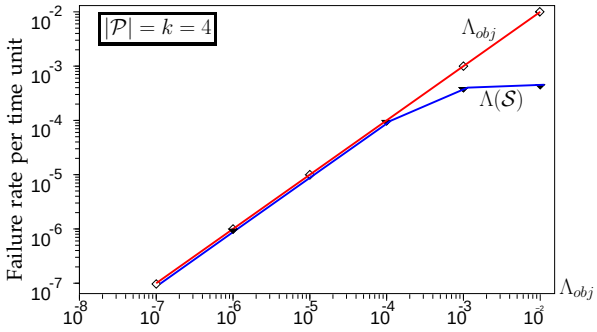

Fig. 14. Variation of the obtained GSFR $\Lambda(S)$ w.r.t. Λ_{obj}.

6.3 Variation of the replication factor w.r.t. Λ_{obj}

Figures 15 and 16 show the average replication factor produced by BSH w.r.t. the objective Λ_{obj}, averaged over the 50 $\mathcal{A}lg$ graphs. Each point is the average replication factor obtained for *one* $\mathcal{A}lg$ graph, and the curve passes through the average value of all the points obtained with a given Λ_{obj}. When Λ_{obj} is within $[10^{-3}, 10^{-2}]$, the replication factor is almost equal to 1, which is consistent with the remark made above concerning the GSFR (the replication factor can never be below 1 since at least one replica of each operation must be scheduled). As we can see, the replication factor grows almost linearly when Λ_{obj} decreases from 10^{-3} to 10^{-7}.

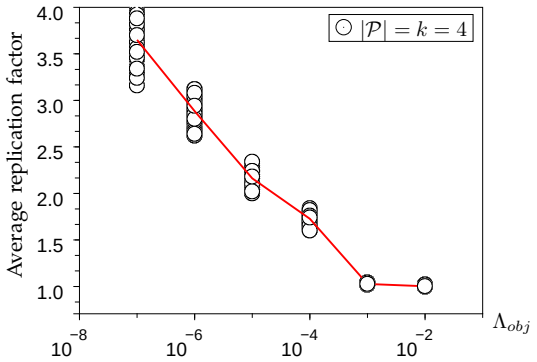

Fig. 15. Variation of the average replication factor w.r.t. Λ_{obj} on a 4 processors architecture.

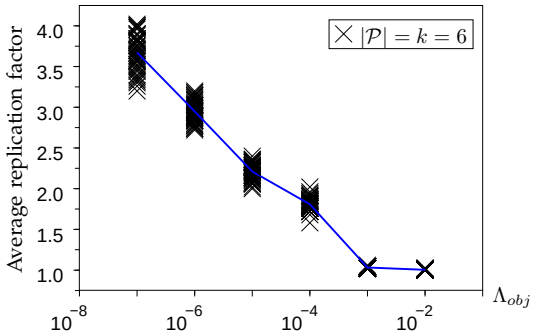

Fig. 16. Variation of the average replication factor w.r.t. Λ_{obj} on a 6 processors architecture.

2. The Communication-to-Computation Ratio is the ratio between the average WCCT (over all the data-dependencies) and the average WCET (over all the operations).

Also, we can see that the two curves are not very different: the average replication factor is not influenced by the size of the target architecture, but only by the objective Λ_{obj} and the individual failure rates per time unit of the architecture's hardware components.

Finally, theoretically, the upper-bound k has an impact on the schedules produced by BSH, because the replication factor of all tasks will be less than or equal to k. But, according to Figures 15 and 16, even when the desired GSFR is as strict as 10^{-7}, the replication factor remains smaller than 4. In conclusion, limiting k has an impact on the complexity of BSH but not on the quality of the produced schedules, unless one limits k to a value incompatible with the desired GSFR. Our simulations tell what value of k is needed w.r.t. the desired GSFR.

6.4 Variation of the exact replication factor w.r.t. Λ_{obj}

For Figures 17 and 18, we have chosen one particular Alg graph of 50 operations. Each operation is numbered from 1 to 50. We have run BSH on this graph with $\Lambda_{obj} = 10^{-5}$ and drawn the exact replication factor of each operation. Both figures correspond to a fully connected architecture, respectively with 4 and 6 processors (whose individual failure rates per time units are respectively given in Table 1. In the 4 processors case, the average replication factor is equal to 2.18, while in the 6 processors case, it is equal to 2.22.

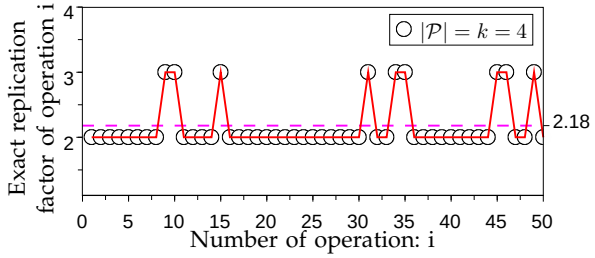

Fig. 17. Exact replication factor of each operation for $\Lambda_{obj} = 10^{-5}$ on a 4 processors architecture.

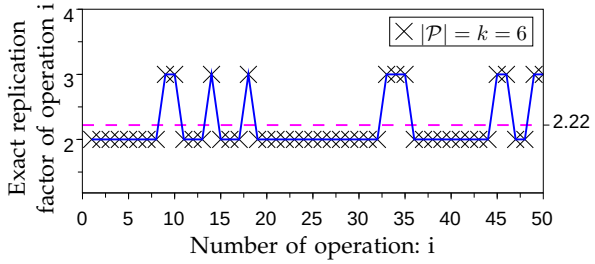

Fig. 18. Exact replication factor of each operation for $\Lambda_{obj} = 10^{-5}$ on a 6 processors architecture.

The important thing to note in Figures 17 and 18 is that the exact replication factors are evenly distributed over the average. Indeed, the standard deviation is only 0.384 for Figure 17 (resp. 0.414 for Figure 18). Furthermore, there is no bad funnel effect.

Finally, the fact that the average replication factor (2.18 for Figure 17 and 2.22 for Figure 18) does not depend on the size of the target architecture is consistent with the observation made in Section 6.3.

6.5 Variation of the average replication factor w.r.t. the processors' failure rate

We are also interested in the average replication factor of the operations scheduled on each processor of the architecture (noted $AORP$). Recall that, for an operation X, $\Omega(\mathrm{X})$ is the set of processors that execute X, and conversely $\Omega^{-1}(\mathrm{P})$ is the set of operations placed onto P. Hence we have:

$$AORP(\mathrm{P}) = \frac{\sum_{\mathrm{X} \in \Omega^{-1}(\mathrm{P})} |\Omega(\mathrm{X})|}{|\Omega^{-1}(\mathrm{P})|} \qquad (19)$$

The $AORP$ is very important to assess the quality of a (length,reliability) bicriteria scheduling algorithm, because it shows how it is related to the failure rate per time unit of each processor. We intuitively expect that, if $\lambda_{\mathrm{P}} < \lambda_{\mathrm{P}'}$, then $AORP(\mathrm{P}) < AORP(\mathrm{P}')$, that is, operations scheduled on more reliable processors should be replicated less. This is consistent because an operation scheduled onto a very reliable processor (i.e., whose failure rate is smaller than Λ_{obj}) does not need to be replicated, while an operation scheduled onto an unreliable processor (i.e., whose failure rate is greater than Λ_{obj}) must be replicated several times to satisfy the Λ_{obj} constraint.

However, when computing the $AORP$, we have to be careful of the values of the failure rate of the communication links. Indeed, these values have a direct influence on the replication factor of each operation (see Section 5.1 and in particular Figure 12). For this reason, the simulation below (Figure 19) is run with three values of the failure rate of the communication media λ_m, different from those given in Table 1: these new values of λ_m are chosen such that the replicas of the data dependencies on the communication media have a reduced influence on the replication factor of the operations.

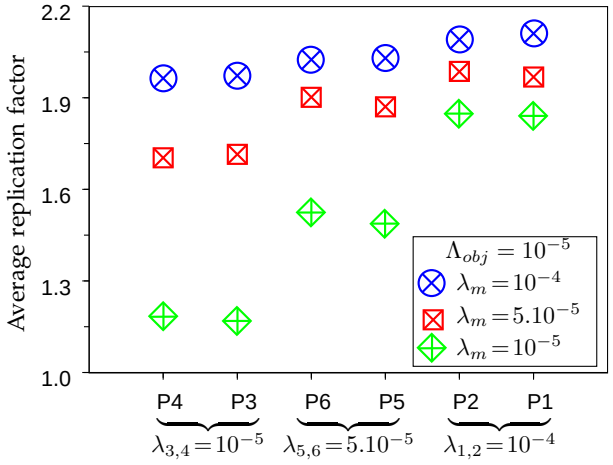

Fig. 19. Average replication factor for each processor.

Figure 19 shows the $AORP$ of each processor P1 to P6 in the architecture with 6 processors, for a value of Λ_{obj} equal to 10^{-5} and for three distinct values of λ_m, namely 10^{-4}, 5.10^{-5} and 10^{-5}. It shows very clearly that $AORP(\mathrm{P})$ is directly related to the failure rate per

time unit of P, which demonstrates that BSH works remarkably well: in the three cases, the more reliable the processor is, the less its average replication factor becomes: $AORP(\mathrm{P5},6) < AORP(\mathrm{P3},4) < AORP(\mathrm{P1},2)$. This order relationship is clearer in the $\lambda_m = 10^{-5}$ case because, in that case, the communication media are more reliable than the processors (actually, they are as reliable as the most reliable processors P5 and P6), hence they have no influence at all on the replication factors of the operations.

6.6 Variation of the $\mathtt{C_{max}}$ overhead w.r.t. Λ_{obj}

Figure 20 shows the $\mathtt{C_{max}}$ overhead of the schedules produced by BSH w.r.t. the objective Λ_{obj}, averaged over the 50 $\mathcal{A}lg$ graphs, computed by Equation (20):

$$\frac{\mathtt{C_{max}}(BSH) - \mathtt{C_{max}}(BSH \text{ without repl.})}{\mathtt{C_{max}}(BSH \text{ without repl.})} \times 100 \qquad (20)$$

where "$\mathtt{C_{max}}(BSH$ without repl.)" denotes the average $\mathtt{C_{max}}$ obtained by a modified BSH that does not replicate the tasks. This figure shows the compromise in terms of $\mathtt{C_{max}}$ that the user has to pay in order to gain one or several orders of magnitude of the failure rate. We vary two parameters: the number of processors in the architecture $|\mathcal{P}|$, and the failure rate per time unit of the communication links λ_m. Three curves are drawn, respectively for $\mathcal{P}$ =4 and λ_m=10^{-3}, for $\mathcal{P}$ =6 and λ_m=10^{-3}, and for $\mathcal{P}$ =4 and λ_m=10^{-4}.

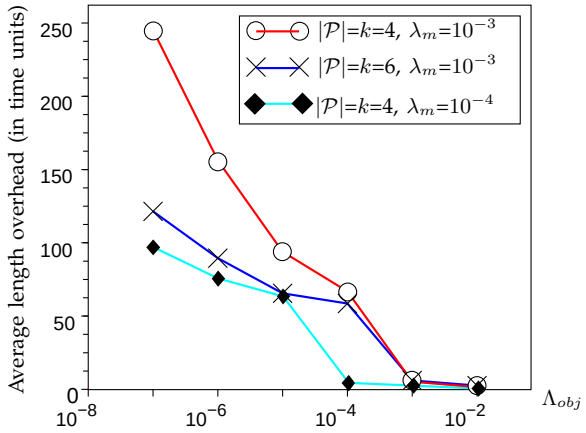

Fig. 20. Variation of the $\mathtt{C_{max}}$ overhead w.r.t. Λ_{obj} and λ_m.

The $\mathtt{C_{max}}$ overhead grows when Λ_{obj} decreases, because the replication factor increases and so does the number of replicas. Another reason for the overhead is that the insertion of routing operations (necessary to yield serial-parallel RBD) increases the $\mathtt{C_{max}}$. Also, the overhead is less when the architecture has more processors, because in that case, the parallelism available in the architecture is greater, hence each processor must execute a smaller number of replicas (even though the average replication factor is roughly the same). Finally, when the communication links become more reliable (i.e., λ_m decreases), the overhead decreases: this is because the operations need to be less replicated to achieve the desired Λ_{obj}.

6.7 Average $\mathtt{C_{max}}$ overhead due to the routing operations w.r.t. λ_m

Table 2 shows the average $\mathtt{C_{max}}$ overhead due to the routing operations. For each $\mathcal{A}lg$ graph and each value of Λ_{obj}, we have computed two static schedules: the first one with BSH (that is, with routing operations), and the second one by removing the routing operations but keeping the same assignment choices of the operations to the processors as in the first schedule. The overhead is then computed by Equation (21):

$$\frac{\mathtt{C_{max}}(BSH) - \mathtt{C_{max}}(BSH \text{ without routing})}{\mathtt{C_{max}}(BSH \text{ without routing})} \times 100 \qquad (21)$$

For the two architectures (with 4 and 6 processors), we have averaged these overheads over 50 $\mathcal{A}lg$ graphs. We have taken three distinct values of the failure rate per time units of the communication media λ_m, namely 10^{-3}, 10^{-4} and 10^{-5}, and we have averaged the overhead over the six usual values of Λ_{obj}, from 10^{-2} to 10^{-7}. Table 2 shows that the overhead due to the routing operations is very reasonable: it varies between $-4.12\,\%$ (an actual gain!) and $+9.96\,\%$.

λ_m	10^{-3}	10^{-4}	10^{-5}
$\lvert\mathcal{P}\rvert = k = 4$	$-4.12\,\%$	$+2.43\,\%$	$+4.09\,\%$
$\lvert\mathcal{P}\rvert = k = 6$	$+2.44\,\%$	$+8.47\,\%$	$+9.96\,\%$

TABLE 2
Average $\mathtt{C_{max}}$ overhead due to the routing operations.

λ_m	10^{-3}	10^{-4}	10^{-5}
$\lvert\mathcal{P}\rvert = k = 4$	2.07	1.50	1.33
$\lvert\mathcal{P}\rvert = k = 6$	2.10	1.52	1.35

TABLE 3
Average replication factor for the schedules with routing operations.

We can see that it decreases when the communication links become less reliable: this is because the average replication factor of the operations increases (see Table 3); hence there is *more locality* in the computations; hence there are more routing operations scheduled on the same processor of either the source or the destination operation of the data-dependency, resulting in fewer inter-processor communications and less $\mathtt{C_{max}}$ overhead. Also, we can see that the overhead is greater for the architecture with 6 processors than for the one with 4 processors: this is because there is more concurrency available to schedule the communications in parallel in a fully connected 6 processor architecture (15 communication links) than in a 4 processor one (6 communication links): indeed, recall that the absence of routing operations means more data-dependencies in parallel (see Figure 6). Finally, the average overhead is only $+3.88\,\%$, which is very reasonable.

6.8 Example of a (length,GSFR) Pareto curve

Figure 21 is an example of a Pareto curve generated by running BSH with 45 different values of Λ_{obj} (from 10^{-7} to 9.10^{-3}) on a given instance of the problem, i.e., a given $\mathcal{A}lg$ of 100 tasks and a given $\mathcal{A}rc$ graph of 10 processors, with the maximal degree of processor combinations set to 7. Among the 10 processors of $\mathcal{A}rc$, 5 had a failure rate per time unit equal to 10^{-4}, the 5 others 10^{-5}; the failure rate par time unit of all the links was equal to 5.10^{-4}. The red solid line connects only the non-dominated points. From this Pareto curve, the user can then chose the solution that best suits her/his requirements in terms of reliability and $\mathtt{C_{max}}$. Finally, the time necessary for BSH to produce these 45 points was 18 528 seconds ($\sim$ 5 hours and 8 minutes) on an Intel Pentium M 740, 1.73 GHz with 1 GO of memory. If we set the maximal degree of processor combinations to 4 instead of 7, we obtain almost exactly the same Pareto curve: the absolute $\mathtt{C_{max}}$ difference between the two sets of data, averaged over the 45 points, is only 12 time units; since the $\mathtt{C_{max}}$ varies from 1 220 to 2 692 time units, this difference is negligible. This shows that limiting the maximal degree of processor combinations does not penalize the quality of the schedules produced by BSH. In contrast, the time necessary for BSH to produce these 45 points was only 5 146 seconds ($\sim$ 1 hour and 26 minutes), a gain of a factor 3.6! Yet, those figures indicate that BSH is not suited to architectures with hundreds of processors.

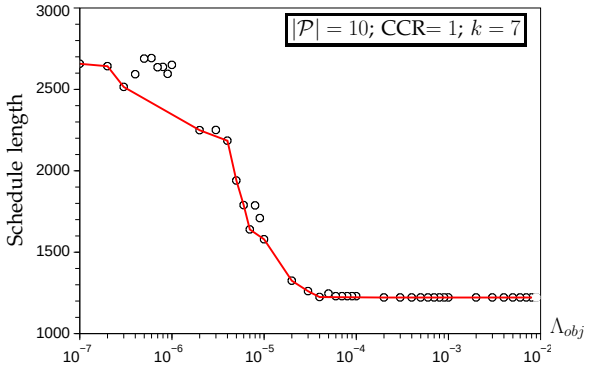

Fig. 21. Example of a (length,GSFR) Pareto curve generated by BSH.

7 Conclusion

We have proposed a new framework for the (length,reliability) bicriteria static multiprocessor scheduling problem. Our first criterion remains the static schedule's length (crucial to assess the system's real-time property). For our second criterion, we consider the global failure rate (GSFR) of the system instead of the usual reliability. The GSFR is the failure rate of the multiprocessor schedule seen as if it were a single operation scheduled onto a single processor; the GSFR is computed from the reliability of the schedule and the total utilization of its hardware components. The reason for this choice is that the GSFR does not depend on the schedule length like the reliability does, due to its computation in the classical reliability model of Shatz and Wang. Thanks to this key independence property we avoid three problems: firstly, the impossibility to control the replication factor of each individual task of the dependency task graph given as a specification, with respect to the desired reliability; secondly, the fact that the length criterion overpowers the reliability criterion; and thirdly, the non-monotonousness of the reliability in function of the schedule. Furthermore, we claim that any bicriteria optimization problem in which the two criteria are not independent one of the other will always suffer from those three problems. 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.

We have proposed a new bicriteria scheduling algorithm, called BSH. It is a greedy list scheduling heuristic that takes as input a task DAG ($\mathcal{A}lg$), a heterogeneous distributed memory architecture ($\mathcal{A}rc$), the worst case execution and communication times of the operations and data-dependencies onto the processors and communication links ($\mathcal{E}xe$), and an upper-bound constraint on the GSFR (Λ_{obj}). BSH returns a static multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$, such that its GSFR is less than Λ_{obj}. The GSFR is improved thanks to the active replication of the operations. At each iteration of BSH, we have to compute the reliability of several partial schedules, one for each replication and assignment choice of the candidate operation onto the processors of $\mathcal{A}rc$; to compute efficiently the reliability of these partial schedules, we have chosen to insert routing operations scheduled between the replicas of any operation that must send some data and the replicas of any operation that must receive this same data: thanks to this choice, the reliability block diagram (RBD) corresponding to the schedule is guaranteed to be serial-parallel, meaning that the reliability can always be computed in linear time.

By invoking iteratively BSH with different values of Λ_{obj}, we are able to produce the Pareto curve for a given instance (i.e., a given $\mathcal{A}lg$, $\mathcal{A}rc$, and $\mathcal{E}xe$), therefore providing the user with the choice among several trade-offs between the execution time and the reliability. Our simulation results indicate what execution time overhead can be expected depending on the failure rate level imposed on a system. More important, our simulation results show that BSH works remarkably well, producing static schedules where the replication factor of the operations decreases when they are scheduled onto more reliable processors. Finally, the overhead incurred by the routing operations is reasonable (only $+3.88\,\%$ on average).

Future work will include the application of our new reliability framework to other task and failure models: for instance, a task with redundant inputs executed on a non fail-silent processor may still execute if one of its predecessor tasks is faulty. Our routing operations could

then be used to vote among the available inputs. This will result in other forms of RBDs, with "N-out-of-M" nodes.

ACKNOWLEDGMENTS

Many thanks to David Powell, Erik Saule, and Denis Trystram for helpful discussions on these topics; to the anonymous reviewers and the associate editor for their constructive remarks; and to Thomas McGowan for his English proficiency.

REFERENCES

[1] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, pages 347–356, Firenze, Italy, June 2004. IEEE, Los Alamitos, CA.

[2] 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.

[3] 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.

[4] M.O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. Reliability*, 35:230–239, August 1986.

[5] 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.

[6] J.-Y. Colin and P. Chretienne. C.P.M. scheduling with small computation delays and task duplication. *Operations Research*, 39(4):680–684, 1991.

[7] 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.

[8] A. Dogan and F. Özgüner. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 48(3):300–314, 2005.

[9] J. Dongara, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Symposium on Parallelism in Algorithms and Architectures, SPAA'07*, pages 280–288, Paris, France, June 2007. ACM, New-York.

[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] J. Gauthier, X. Leduc, and A. Rauzy. Assessment of large automatically generated fault trees by means of binary decision diagrams. *J. of Risk and Reliability*, 221(2):95–105, 2007.

[12] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *7th International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy, May 1999. ACM, New-York.

[13] M. Hakem and F. Butelle. A bi-objective algorithm for scheduling parallel applications on heterogeneous systems subject to failures. In *Rencontres Francophones du Parallélisme, RENPAR'06*, Perpignan, France, October 2006.

[14] C. Hirel, R. Sahner, X. Zang, and K.S. Trivedi. Reliability and performability modeling using Sharpe. In *International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, TOOLS'00*, volume 1786 of *LNCS*, pages 345–349. Springer-Verlag, March 2000.

[15] C.-C. Hsieh and Y.-C. Hsieh. Reliability and cost optimization in distributed computing systems. *Computers and Operations Research*, 30(8):1103–1119, July 2003.

[16] M.A. Iverson. *Dynamic Mapping and Scheduling Algorithms for a Multi-User Heterogeneous Computing Environement*. PhD Thesis, Ohio State University, Columbus (OH), USA, 1999.

[17] P.A. Jensen and M. Bellmore. An algorithm to determine the reliability of a complex system. *IEEE Trans. Reliability*, 18:169–174, November 1969.

[18] S. Kartik and C.S.R. Murthy. Improved task allocation algorithms to maximize reliability of redundant distributed computing systems. *IEEE Trans. Reliability*, 44(4):575–586, December 1995.

[19] 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.

[20] D. Lloyd and M. Lipow. *Reliability: Management, Methods, and Mathematics*, chapter 9. Prentice-Hall, 1962.

[21] P. Pop, K. Poulsen, and V. Izosimov. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *CODES-ISSS'07*, Salzburg, Austria, October 2007. ACM, New-York.

[22] X. Qin and H. Jiang. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Computing*, 32(5-6):331–356, June 2006.

[23] 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.

[24] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection constraint heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[25] 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.

[26] 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.

[27] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag, 2006.

[28] 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.