Revisiting the bicriteria (length,reliability) multiprocessor static scheduling problem

Alain Girault, Hamoudi Kalla

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Revisiting the bicriteria (length,reliability) multiprocessor static scheduling problem

Alain Girault — Hamoudi Kalla

N° ????

October 2007

Thème COM

ISSN 0249-6399

INRIA
RHÔNE-ALPES

Revisiting the bicriteria (length,reliability) multiprocessor static scheduling problem

Alain Girault*, Hamoudi Kalla†

Thème COM — Systèmes communicants
Projets POP ART

Rapport de recherche n° ???? — October 2007 — 36 pages

Abstract: Our starting point is a dependency task graph and an heterogeneous distributed memory target architecture. We revisit the well studied problem of bicriteria (length,reliability) multiprocessor static scheduling of this task graph onto this architecture. Our first criteria remains the static schedule's length: this is crucial to assess the system's real-time property. For our second criteria, 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 reliability model of Shatz and Wang). 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 application 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.

Key-words: Bicriteria optimization, reliability, Pareto optima, static multiprocessor scheduling, reliability block diagrams.

* Alain Girault is with INRIA and Grenoble University (POP ART project-team and LIG laboratory), France, Email: Alain.Girault@inria.fr

† Hamoudi Kalla is with the University of Batna, Algeria, Email: Hamoudi.Kalla@inrialpes.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Le problème de l'ordonnancement statique multiprocesseurs bicritère (longueur,fiabilité) revisité

Résumé : Notre point de départ est un graphe de dépendences de tâches et une architecture cible à mémoire répartie. Nous revisitons le problème classique de l'ordonnancement statique multiprocesseurs bicritère (longueur,fiabilité) de ce graphe de tâche sur cette architecture. Notre première critère reste la longueur de l'ordonnancement statique: ceci est crucial afin d'établir la propriété temps-réel du système. Pour notre second critère, nous considérons le taux de défaillance du système global, vu comme si le système dans son entier était une unique tâche ordonnancée sur un unique processeur, à la place de l'usuelle fiabilité du système, parce qu'il ne dépend pas de la longueur de l'ordonnancement comme c'est le cas de la fiabilité (en raison de son calcul à l'aide du modèle de fiabilité classique de Shatz et Wang). Ainsi, nous contrôlons beaucoup mieux le niveau de réplication de chaque tâche individuelle du graphe de dépendences donné comme spécification par rapport au taux de défaillance désiré.

Afin de résoudre ce problème d'optimisation bicritère, nous prenons la taux de défaillance comme une contrainte, et nous minimisons la longueur de l'ordonnancement. Nous sommes ainsi capables de produire, pour un graphe de dépendences donné et pour une architecture multiprocesseurs donnée, une courbe de Pareto de solutions non-dominées, parmi lesquelles l'utilisateur peut choisir le compromis qui correspond le mieux à ses exigences.

Mots-clés : Optimisation bicritère, fiabilité, optima de Pareto, ordonnancement statique multiprocesseurs, bloc-diagrammes de fiabilité.

1 Introduction

Bicriteria (length,reliability) multiprocessor scheduling has recently attracted a lot of attention [7, 1, 8, 11, 22, 20]. 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).

We use the widely accepted reliability model of Shatz and Wang [23], where each hardware component (processor or communication link) 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). 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 the schedule length. In other words, adding more replicas improves the reliability but penalizes the 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. This is counter-intuitive though, since this means that replication is bad for reliability! 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 a consequence, 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 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 cases, applying brutally the Shatz and Wang reliability model yields very low reliabilities due to very long execution times (the same remark applies also to very long schedules). Hence

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

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 life; that is, 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. Figure 1 shows a set of solutions for a bicriteria minimization problem: the points x^1, x^2, x^3, x^4, and x^5 are **Pareto optima** [30]; 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 multi-criteria) [30]:

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

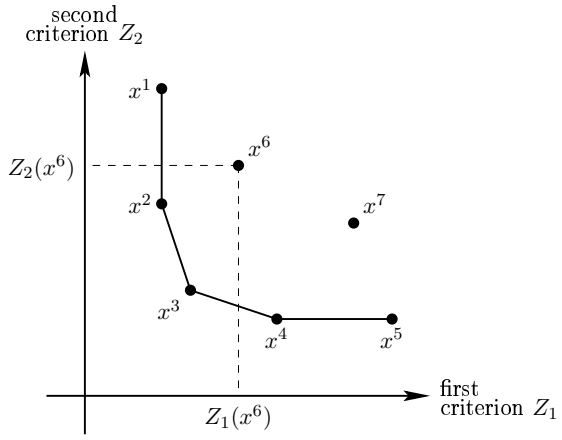

Figure 1: Pareto optima and Pareto curve for a bicriteria minimization problem.

Our second contribution belongs to the second case: we propose a new static scheduling algorithm that takes as a *constraint* a given failure rate per time unit, and that attempts

to *minimize* the schedule length on an heterogeneous architecture, by using a smart cost function. By executing this algorithm with several failure rates, it is possible to obtain the Pareto curve for a given instance of the problem (i.e., a given algorithm graph and a given distributed architecture). 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 most advanced are [7, 1, 8, 22, 11] because, in contrast with many other results, they do not assume the communication network to be acyclic. There are also the works of Srinivasan and Jha [28], but they only aim at maximizing the reliability and not at minimizing the length of the schedule.

Computing the reliability of a schedule requires to compute, for each data-dependency from operation X to operation Y (noted X$\triangleright$Y), the probability that there exists an operational path from the source processor (which executes X) to the destination processor (which executes Y). This problem, known as the **terminal pair problem** [3], is NP-hard in the case of a general network. For this reason, numerous works have restricted themselves to acyclic networks [23, 24, 15, 16, 13, 12], because the time necessary to compute the probability that a path is operational becomes linear in the length of the path.

The RDLS algorithm ("Reliable Dynamic Level Scheduling") proposed by Dogan and Özgüner in [7] is an extension of the Dynamic Level Scheduling (DLS) algorithm of Sih and Lee [25]. 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 $\langle \text{operation}, \text{processor} \rangle$ and chooses to place on processor p the operation o such that the cost function $DL(o,p)$ be maximal. Dogan and Özgüner add an additional term to the DL cost function so as to take into account the reliability of the scheduling (under the classical reliability model of Shatz and Wang). Hence the two criteria, length and reliability, are combined to become 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 they do not replicate operations nor data-dependencies.

In [8], Dogan and Özgüner propose 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.

In [11], Hakem and Butelle have proposed an algorithm very similar to RDLS: indeed, like in [7], the two criteria are aggregated into a single bi-objective cost function, the reliability model is the one of Shatz and Wang, and the tasks are not replicated to increase the reliability. The function $f(t,p)$ denotes the finish time of the task t when it is scheduled on the processor p. The function $\sigma(t,p)$ denotes the reliability cost when the task t is scheduled on the processor p. This bi-objective 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 criteria from dominating the other one, like in [1]. Compared to RDLS, Hakem and Butelle claim to have better results in reliability but worse in schedule length.

The eFCRD algorithm ("Efficient Fault-Tolerant Reliability Cost-Driven Algorithm") proposed by Qin et al. in [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 exactly one processor failure is 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.

Pop et al. have addressed the tricriteria optimization problem (length,reliability,energy) [20]. Both the length and the reliability are taken as a constraint, respectively with a given upper and lower bound. 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.

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 p. For each choice $\langle o, p \rangle$, we compute the induced variation in length Δl 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 $\langle \Delta l, \Delta r \rangle$ 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 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 [1,22] replicate the tasks to increase the reliability. In contrast, [7, 8, 11, 20] cannot increase the reliability of the obtained system above the reliability level of the hardware platform the system is executed on. The approach we present in this article uses 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 [7, 1, 8, 11, 22, 20] 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. The approach we present in Sections 3, 4, and 5 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, in [23, 24, 15, 16, 13, 12], the communication network must be acyclic, which is an important restriction. Besides, in [23, 24, 15, 16, 12], the system to be distributed and scheduled consists of a set of software modules that communicate between them, but these communications do not influence the order of execution of the modules, i.e., they are not data-dependencies; rather they are assumed to be bi-directional communications and are abstracted as an **inter-module communication time** [5]. 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 [15, 16], this exhaustive exploration remains very costly, hence they are only capable of treating very small size problems (less than 8 modules). Finally, [23, 15, 12] treat the particular case of redundant distributed systems, where each computing node consists of n identical processors, and each connection between two computing nodes consists of m identical communication links. More specifically, they consider systems with a redundancy level of 2 or 3, but without software redundancy.

3 Preliminaries

3.1 Algorithm graph

An algorithm graph $\mathcal{A}lg$ is an **acyclic oriented graph** $(\mathcal{O}, \mathcal{D})$. Its nodes (the set $\mathcal{O}$) 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$\triangleright$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$\triangleright$Y, and Y is its **destination**. The set of all predecessors of X is noted $pred$(X). The set of all successors of X is noted $succ$(X). The graph $\mathcal{A}lg$ induces a **partial order relationship** between the operations; this partial order represents the potential parallelism of the algorithm specification.

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.

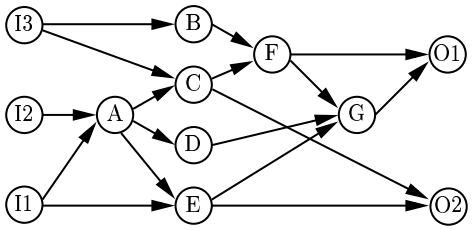

Figure 2: An example of algorithm graph $\mathcal{A}lg$: I1, I2, and I3 are input operations, O1 and O2 are output operations, A-G are regular operations.

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 while $\mathcal{L}$ is the set of communication links. A **processor** is composed of a computing unit, which allows it to execute operations, and one or more communication units, which allow it 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 $\mathcal{A}rc$ connects necessarily one processor and one communication link.

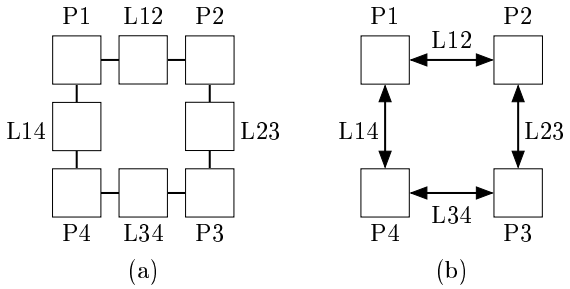

Figure 3: An example of an architecture graph $\mathcal{A}rc$: (a) its bipartite representation; (b) its classical representation.

A **path** between processors P1 and Pn is a sequence P1-A1-L1-A2-P2-A3-L2-A4-$\cdots$-Pn, such that Pi$\in\mathcal{P}$, Li$\in\mathcal{L}$, Ai$\in\mathcal{A}$, A1=(P1,L1), A2=(L1,P2), A3=(P2,L2), A4=(L2,P3), and so on. The graph $\mathcal{A}rc$ is **connected** if there exists a path between any two nodes; this notion is usually defined for non-bipartite graphs, but it applies also to bipartite ones; what matters here is that in a connected $\mathcal{A}rc$ graph, there exists a path between any two processors (and not only between any two nodes). By analogy with non-bipartite graphs, the graph $\mathcal{A}rc$ is **complete** if there exists a communication link between any two processors (and not only

an edge between any two nodes). 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.

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. The WCET analysis is the topic of much work (see [21,18] for surveys). 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 [6] or with caches and pipelines [29]. In particular, it has been applied to the most critical embedded system, namely the AIRBUS A380 avionics software running on the MOTOROLA MPC755 processor [9,27].

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 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 $\mathcal{A}rc$, composed on one hand of a spatial allocation function, and on the other hand of a temporal allocation function. The **spatial allocation function** Σ 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.

The **temporal allocation function** Ω gives the starting date of each operation (resp. each data-dependency) on its processor (resp. its communication link). In general, for two given graphs $\mathcal{A}lg$ and $\mathcal{A}rc$, there exist several possible schedules.

A schedule is **static** if the function Ω is static. This means that all the scheduling decisions have been made off-line. Otherwise the schedule is **dynamic**.

A static schedule is **without replication** if for each operation X (and for each data-dependency), the set $\Sigma(\mathrm{X})$ is a singleton, or in other words, $|\Sigma(\mathrm{X})| = 1$. In contrast, a schedule is **with (active) replication** if for some operation X (or some data-dependency), $|\Sigma(\mathrm{X})| \geq 2$. The number $|\Sigma(\mathrm{X})|$ is called the **replication factor** of X. In a schedule with replication, a given operation can start its execution as soon as it has received all its input data at least from one active replica of each of its predecessors. In other words, it does not need to wait for the reception of its input data sent by all the replicas of its predecessors.

A schedule is **partial** if not all the operations of $\mathcal{A}lg$ have been scheduled, but of course all the operations that are scheduled are such that all their predecessors are also scheduled.

Finally, the **length** or **makespan** of a static 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 $\mathtt{C}_{\mathtt{max}}$.

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 (and similarly for the communication links); this is known as the "hot" failure model. According to this failure model, the occurrence of failures on a processor P (resp. a communication link L) follows a Poisson law with a constant parameter λ, called its **failure rate per time unit**. Moreover, failure occurrences are statistically independent events. These assumptions are the same as those made in [23, 24, 15, 16, 13, 12, 22, 7, 1, 8, 20]. Note also that transient failures are the most common failures in modern embedded systems.

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:

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

Modern fail-silent processors can have a failure rate around $10^{-6}/hr$. For instance, the probability that a processor, whose failure rate is equal to $10^{-6}/hr$, be operational during 2 hours is $e^{-10^{-6}*2} \simeq 0.999998$. Concerning its probability of failure during the same duration, it is equal to $1 - e^{-10^{-6}*2} \simeq 0.000002$.

3.6 Computing the reliability of a schedule

Definitions **Reliability Block-Diagrams** (RBD) have been introduced to compute the reliability of a system [19]. Formally, an RBD is an **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. In any RBD, there are two particular blocks: its **source** S and its **destination** D. An RBD represents a system and is used to compute its reliability: 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 formula (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 [17].

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. Also, an RBD is **parallel** if all its blocks are in parallel. Finally, an RBD is **serial-parallel** if its consists of a sequence of parallel portions. In those last two cases, the computation of the reliability is also *linear* in the size of the RBD.

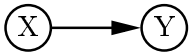

Figure 4: A simple $\mathcal{A}lg$ graph.

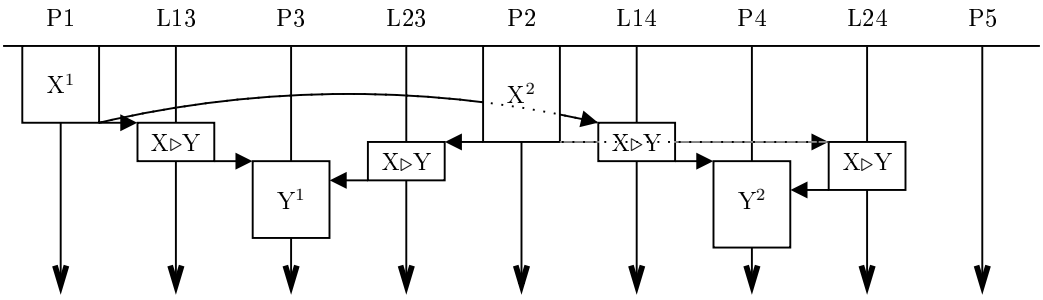

Figure 5: A simple schedule without routing.

As an example, consider the simple $\mathcal{A}lg$ graph of Figure 4. A possible scheduling with replication of this $\mathcal{A}lg$ graph is show in Figure 5, 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 6, with source S and destination D. This RBD has no particular form

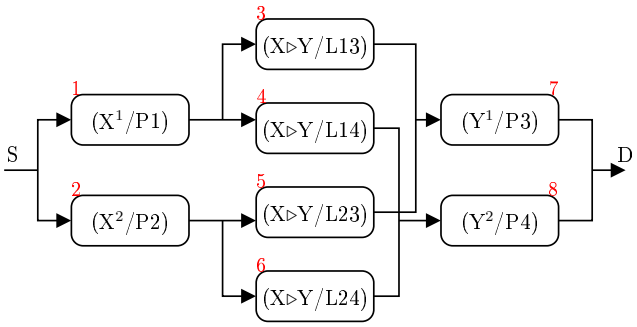

Figure 6: The RBD of the schedule of Figure 5.

Minimal cut sets method There are two main solutions to overcome this problem. First, one can use the **minimal cut sets method** [14, 4]. This method is used by commercial tools such as Reliability Workbench from ISOGRAPH [2] or Item Toolkit from ITEM [3]. 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:

$$R(\mathcal{C}_i) = 1 - \prod_{(o,c)\in\mathcal{C}_i} \Big(1 - R(o,c)\Big) \tag{2}$$

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:

$$R(S) \geq R^-(S) = \prod_{i=1}^{n} \left(1 - \prod_{(o,c)\in\mathcal{C}_i} \Big(1 - R(o,c)\Big)\right) \tag{3}$$

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,

[2] ISOGRAPH: http://www.isograph-software.com.htm.
[3] ITEM: http://www.itemuk.mcmail.com/rbd.html

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 6 has 11 minimal cut sets: $\{1,2\}$, $\{3,4,5,6\}$, $\{7,8\}$, $\{1,5,6\}$, $\{2,3,4\}$, $\{3,5,8\}$, $\{4,6,7\}$, $\{1,6,7\}$, $\{1,5,8\}$, $\{2,4,7\}$, and $\{2,3,8\}$. As a consequence, 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 proposed in [1].

Serial-parallel schedule method For this reason, we propose to produce schedules in such a way that the corresponding RBD will be, by construction, serial-parallel. This allows us to compute its reliability in a linear time w.r.t. the size of S. The drawback is that the schedules we produce have a greater $\mathtt{C_{max}}$, that is, a higher overhead; however, we will see in Section 6.7 that the additional overhead incurred by this design choice is reasonable. 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.

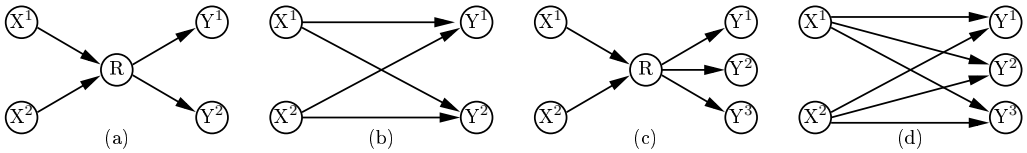

Figure 7: (a) A transformed *Alg* graph with replication and with a routing operation R when X and Y are replicated twice; (b) Same as (a) without the routing operation; (c) Same as (a) when Y is replicated thrice; (d) Same as (c) without the routing operation.

Concerning the scheduling algorithm, this method involves inserting an additional **routing operation**, whose WCET is equal to 0 time unit, between the data-dependencies coming out of the source operation and the data-dependencies going towards the destination operation. The reason of this choice is that the RBD of such schedules with routing operations are guaranteed to be serial-parallel, hence making the computation of the reliability linear instead of exponential in the size of the RBD (see Section 3.6). Figure 7(a) shows the graph transformation that occurs from the *Alg* graph of Figure 4 when X is replicated twice (X^1 and X^2) and Y is replicated thrice (Y^1, Y^2, and Y^3). In such a case, 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 7(a) has four communications. In comparison, when no routing operation is inserted, Figure 7(b) shows the transformed *Alg* graph: it also has four communications, but they are *concurrent*, while in Figure 7(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 7(c) and 7(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 7(c), 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.

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 $\Lambda(S) = \dfrac{-\log R(S)}{U(S)}$, where U is computed as $U(S) = \sum_{o_i \in S} \mathcal{E}xe(o_i)$.*

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.

4.2 A schedule without replication

Consider again the very simple $\mathcal{A}lg$ graph of Figure 4, scheduled onto an $\mathcal{A}rc$ graph composed of two processors P1 and P2, connected by a point-to-point communication link L12. Figure 8 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 the WCET (resp. the WCCT) of that operation onto its processor (resp. its communication link).

Let λ_1 be the failure rate of P1, λ_2 be the failure rate of P2, and λ_{12} be the failure rate of L12. Let t_X^1=WCET(X,P1), t_Y^2=WCET(Y,P2), and t_{XY}^{12}=WCCT(X▷Y,L12). By applying equation (1), we obtain $R(X,P1) = e^{-\lambda_1 t_X^1}$, $R(X \triangleright Y, L12) = e^{-\lambda_{12} t_{XY}^{12}}$, and $R(Y,P2) = e^{-\lambda_2 t_Y^2}$. Figure 9 shows the RBD corresponding to the schedule of Figure 8:

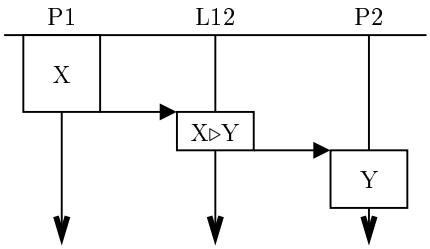

Figure 8: A simple schedule without replication.

Figure 9: The RBD of the schedule of Figure 8.

Since those three blocks are in sequence in the RBD, we have the following global reliability:

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

By applying the formula of Definition 4.1, the GSFR of this system is:

$$\Lambda = \frac{-\log R}{U} = \frac{\lambda_1 t_X^1 + \lambda_{12} t_{XY}^{12} + \lambda_2 t_Y^2}{t_X^1 + t_{XY}^{12} + t_Y^2} \tag{4}$$

In the particular case where the architecture is homogeneous w.r.t. the reliability, $\lambda_1 = \lambda_2 = \lambda_{12}$, hence $\Lambda = \lambda_1$. This result is perfectly consistent with our definition of the GSFR (Definition 4.1).

4.3 A schedule with replication

We now consider a schedule with replication of the same $\mathcal{A}lg$ graph, X→Y, scheduled onto an $\mathcal{A}rc$ graph composed of five processors P1 to P5, completely connected by point-to-point

communication links. Figure 10 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, the scheduling algorithm will make its best to group some operations on the same processor so as to avoid inter-processor communications. The RBD corresponding to this schedule is shown in Figure 11, with source S and destination D.

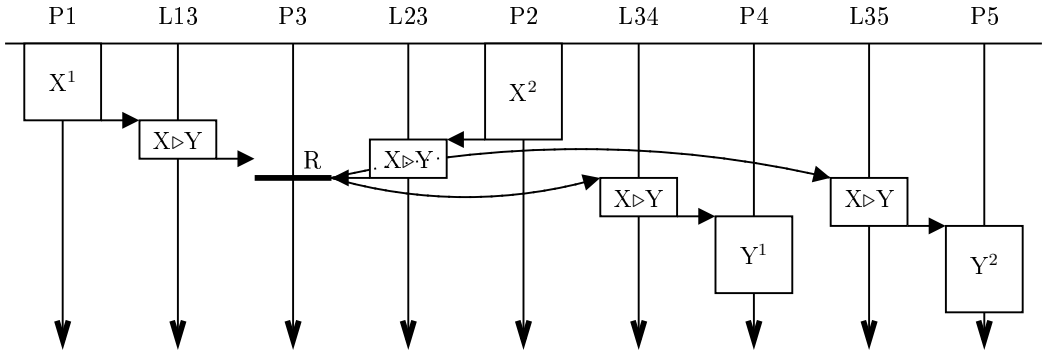

Figure 10: A simple schedule with replication.

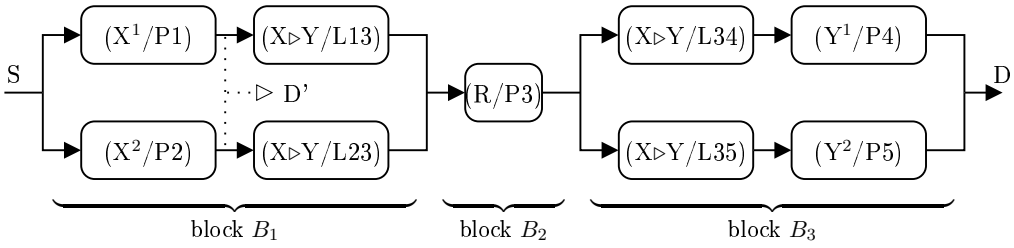

Figure 11: The RBD of the schedule of Figure 10.

First, we consider the partial schedule S' composed only of the first two operations X^1 and X^2 (i.e., the two replicas of X) in the schedule of Figure 10. The corresponding RBD is also shown in Figure 11, but with source S and destination D' (it's actually a sub-graph). 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., $t_X^2 = WCET(X, P2)$):

$$\begin{aligned} R(S') &= 1 - \big(1 - R(X^1, P1)\big)\big(1 - R(X^2, P2)\big) \\ &= 1 - \left(1 - e^{-\lambda_1 t_X^1}\right)\left(1 - e^{-\lambda_2 t_X^2}\right) \end{aligned} \tag{5}$$

For the sake of the example, let us take the following numbers: $\lambda_1 = \lambda_2 = 10^{-5}$ per time unit and $t_X^1 = t_X^2 = 5$ time units. We thus obtain:

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

To compute the GSFR $\Lambda(S')$, we apply the formula of Definition 4.1:

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

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:

$$R(S') \simeq 1 - (\lambda_1 t_X^1)(\lambda_2 t_X^2) \tag{7}$$

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

$$\Lambda(S') \simeq \frac{\lambda_1 t_X^1 \lambda_2 t_X^2}{U(S')} = \frac{\lambda_1 t_X^1 \lambda_2 t_X^2}{t_X^1 + t_X^2} \tag{8}$$

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 WCET of X^i on Pi is t_X^i. This yields:

$$\Lambda(S') \simeq \frac{\prod_{i=1}^{n} \lambda_i t_X^i}{\sum_{i=1}^{n} t_X^i} \tag{9}$$

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 product of the execution times. This is exactly what one expects when adding active replicas to a schedule.

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

$$\begin{array}{l} R(B_1) = 1 - \left(1 - e^{-(\lambda_1 t_X^1 + \lambda_{13} t_{XY}^{13})}\right)\left(1 - e^{-(\lambda_2 t_X^2 + \lambda_{23} t_{XY}^{23})}\right) \\ R(B_2) = 1 \text{ because the WCET of R is always 0} \\ R(B_3) = 1 - \left(1 - e^{-(\lambda_{34} t_{XY}^{34} + \lambda_4 t_X^4)}\right)\left(1 - e^{-(\lambda_{35} t_{XY}^{35} + \lambda_5 t_X^5)}\right) \end{array}$$

And the reliability of the schedule is $R(S) = R(B_1)R(B_2)R(B_3)$. For the sake of the example, let us take the following numbers: $\forall i, \lambda_i = 10^{-5}$ per time unit, $\forall i, j, \lambda_{ij} = 10^{-4}$

per time unit, $t_X^1 = t_X^2 = 5$ time units, $t_Y^4 = t_Y^5 = 5$ time units, and $\forall i, j, t_{XY}^{ij} = 3$ time units. 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 7), the active replication of X and Y decreases the GSFR, but the addition of the routing operation increases the $\mathtt{C_{max}}$ (it would have been 13 time units instead of 16). Of course, when computing the multiprocessor schedule, our scheduling algorithm will try to group some operations on the same processor so as to avoid inter-processor communications.

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 blocks B_1 and B_2, with respective failure rate λ_1 and λ_2, and respective WCET 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 4, where X is replicated twice and Y is not replicated. This schedule is actually a subset of the one shown in Figure 10, 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 $B_2 = R \cdot X \triangleright Y/L34 \cdot Y/P4$). Hence, by applying

equations (10) and (11), we obtain:

$$\Lambda_{1_1} = \frac{\lambda_1 t_X^1 + \lambda_{13} t_{XY}^{13}}{t_X^1 + t_{XY}^{13}} \qquad T_{1_1} = t_X^1 + t_{XY}^{13}$$

$$\Lambda_{1_2} = \frac{\lambda_2 t_X^2 + \lambda_{23} t_{XY}^{23}}{t_X^2 + t_{XY}^{23}} \qquad T_{1_2} = t_X^2 + t_{XY}^{23}$$

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

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

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

For the sake of the example, let us take the following numbers: $\forall i, \lambda_i = 10^{-5}$ per time unit, $\forall i, j, \lambda_{ij} = 10^{-4}$ per time unit, $t_X^1 = t_X^2 = 5$ time units, $t_Y^4 = 5$ time units, and $\forall i, j, t_{XY}^{ij} = 3$ time units. 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. The order of magnitude of their sum is 10^{-5}, which means that Λ_2 "wins" over Λ_1, but the $\frac{2}{3}$ multiplicative factor allows us to "regain" some part of an order of magnitude.

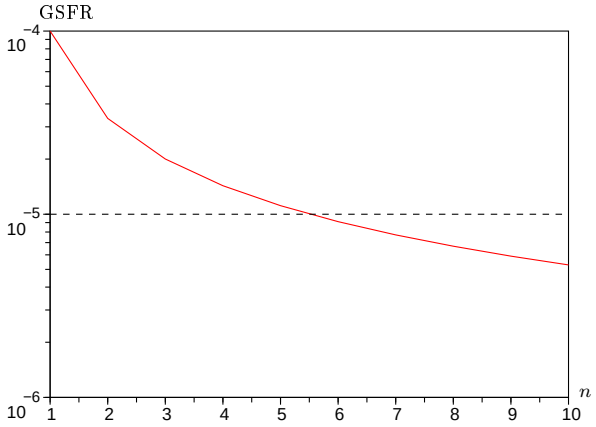

Figure 12: GSFR of a system composed of n blocks in sequence.

It is interesting to generalize this RBD to n blocks in sequence, where the first $n-1$ blocks are each composed of two blocks in parallel (because the corresponding operation or data-dependency is replicated twice), while the last block corresponds to a single operation. Assuming, for the sake of simplicity, that the execution times of all the blocks are identical (and 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 and communication links are identical (and equal to 10^{-4}); we then have $\forall 1 \leq i \leq n-1, \Lambda_i = 2.5\,10^{-8}$ and $\Lambda_n = 10^{-4}$. Depending on n, we thus have the GSFR shown in Figure 12. 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.

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 $\mathcal{A}lg$, and 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 us call $(B_i)_{1\leq i\leq n}$ the n macro-blocks of the serial-parallel RBD. Let us note $\Lambda(B_i)$ the GSFR of B_i and $U(B_i)$ the utilization of B_i. 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, equation (10) gives the following GSFR for the whole RBD:

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

Hence, according to inequality (12), we have $\Lambda \leq \Lambda_{obj}$. □

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

The dependable schedule pressure is a variant of the schedule pressure cost function σ proposed in [26,10], which tries to minimize the length of the critical path of the algorithm graph and 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:

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

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 and data-dependencies are not scheduled yet, we do not know their WCET (resp. WCCT), so we compute the average WCET (resp. WCCT) of each operation (resp. data-dependency) on all processors (resp. communication links).

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

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

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 criteria 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 equation (15):

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

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 $\mathcal{P}_k$. 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 block that contains the replicas of o_i on the processors of $\mathcal{P}_k$ is less than Λ_{obj} (see Figure 11). If all blocks are such that $\Lambda_{block} \leq \Lambda_{obj}$, then $\Lambda^{(n)} \leq \Lambda_{obj}$, thanks to Proposition 5.1.

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 7, these communications will be replicated into the same number of replicas as Y, but the links where they will be scheduled will only be known at the next step of BSH. As a consequence, for an $\mathcal{A}lg$ graph of the form X→Y→Z, we build the RBD of Figure 13, where the future communications are in red/dotted.

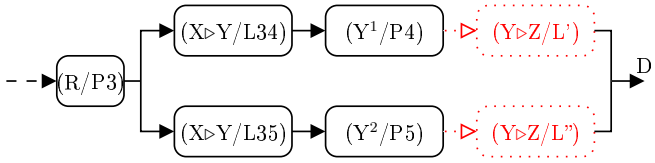

Figure 13: A RBD taking into account the future communications of Y.

Furthermore, when selecting the links to compute the reliability of the blocks corresponding to these future communications (L' and L" in Figure 13), we choose the links having the worse failure rates, in order to guarantee that, whatever the scheduling choice made during the next step $n+1$ of BSH, $\Lambda^{(n+1)}$ will be less than Λ_{obj}.

5.2 Bicriteria scheduling heuristic (BSH)

The BSH scheduling heuristic is shown in Figure 14. Initially, $L^{(0)}_{sched}$ is empty and $L^{(0)}_{cand}$ is the list of operations without any predecessors. At the n-th step, these lists are updated according to the data-dependencies of $\mathcal{A}lg$.

At each step 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 $\langle o_{urg}, \mathcal{P}^{(n)}_{best}(o_{urg})\rangle$. 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 implementations are also replicated and scheduled according to the graph transformations of Figures 7(a) and 7(c). We check at the micro-step ⑤ if the

failure rate objective is satisfied or not. If it is not, the user can change the failure rate objective Λ_{obj} or upgrade the architecture $\mathcal{A}rc$, and re-execute the algorithm. Finally, we update the lists of candidate and scheduled operations at the micro-step ⑥.

Algorithm BSH:
input: $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{E}xe$, and Λ_{obj};
output: a reliable distributed 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:

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

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} and re-execute the algorithm */

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

⑦ $n := n + 1$;

end while
end

Figure 14: The BSH scheduling heuristic.

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.[4] Each of these graphs were then given to BSH with the two heterogeneous $\mathcal{A}rc$ graph having respectively 4 and 6 processors, completely connected. Table 1 gives the individual failure rates per time unit of all the processors and communication links in the 4 processors architecture.

P1,P2	P5,P6	L12,L15,L16,L25,L26,L56
$\lambda_{1,2} = 10^{-4}$	$\lambda_{5,6} = 10^{-5}$	$\lambda_m = 10^{-3}$

Table 1: Failure rates per time unit in the 4 processors architecture.

Table 2 gives the individual failure rates per time unit of the additional processors and communication links in the 6 processors architectures

P3,P4	L13,L14,L23,L24,L34,L35,L36,L45,L46
$\lambda_{3,4} = 5.10^{-5}$	$\lambda_m = 10^{-3}$

Table 2: Failure rates per time unit in the 6 processors architecture.

6.2 Variation of the GSFR in function of Λ_{obj}

Figure 15 shows the actual GSFR obtained by BSH in function of 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 in the interval $[10^{-7}, 10^{-4}]$, $\Lambda(S)$ remains very close to Λ_{obj}. In contrast, when Λ_{obj} is in the interval $[10^{-4}, 10^{-2}]$, then $\Lambda(S)$ is significantly lower than Λ_{obj}. This is because it is not possible to obtain such a low level of failure rate with the processors and communication links of $\mathcal{A}rc$.

[4] The Communication-to-Computation Ratio is the ratio between the average WCTT (over all the data-dependencies) and the average WCET (over all the operations).

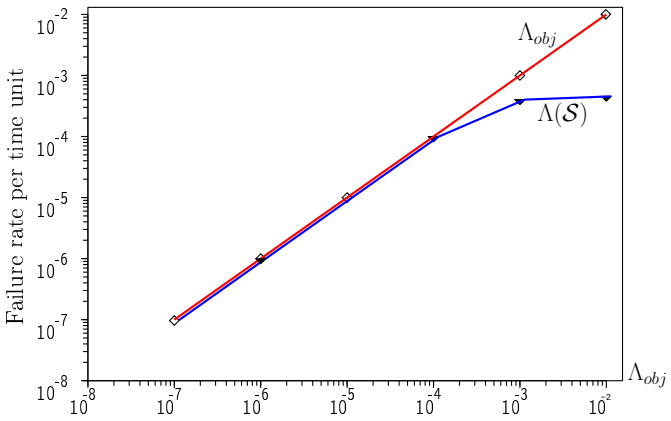

Figure 15: Variation of the obtained GSFR $\Lambda(S)$ in function of Λ_{obj}.

6.3 Variation of the average replication factor in function of Λ_{obj}

Figures 16 and 17 show the average replication factor produced by BSH in function of the objective Λ_{obj}, averaged over the 50 $\mathcal{A}lg$ graphs. Each point indicates the average replication factor obtained for *one* $\mathcal{A}lg$ graph, and the curve passes through the middle of all the points obtained with a given Λ_{obj}. Note that when Λ_{obj} is in the interval $[10^{-4}, 10^{-2}]$, 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^{-6}.

Also, we can see that the two curves are not very different. That is, 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 hardware components.

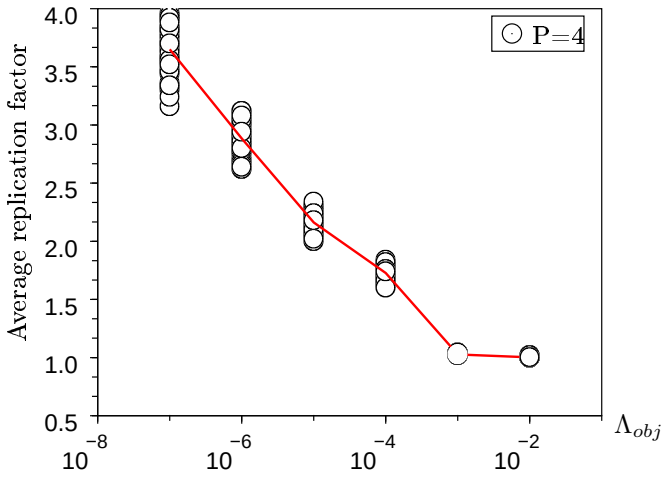

Figure 16: Variation of the average replication factor in function of Λ_{obj} on a 4 processors architecture.

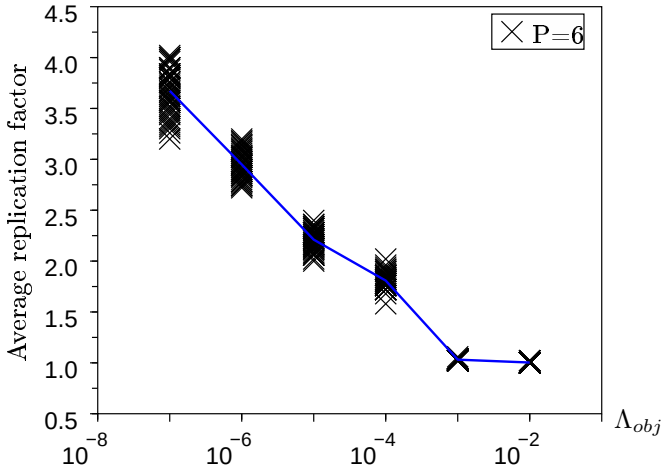

Figure 17: Variation of the average replication factor in function of Λ_{obj} on a 6 processors architecture.

6.4 Variation of the exact replication factor in function of Λ_{obj}

For Figures 18 and 19, we have chosen one particular $\mathcal{A}lg$ 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 Tables 1 and 2). 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.

The important thing to note in Figures 18 and 19 is that the exact replication factors are evenly distributed over the average. Indeed, the standard deviation is only 0.384 for Figure 18 (resp. 0.414 for Figure 19). Furthermore, there is no bad "funnel" effect.

Finally, the fact that the average replication factor (2.18 for Figure 18 and 2.22 for Figure 19) does not depend on the size of the target architecture is consistent with the observation made in Section 6.3 above.

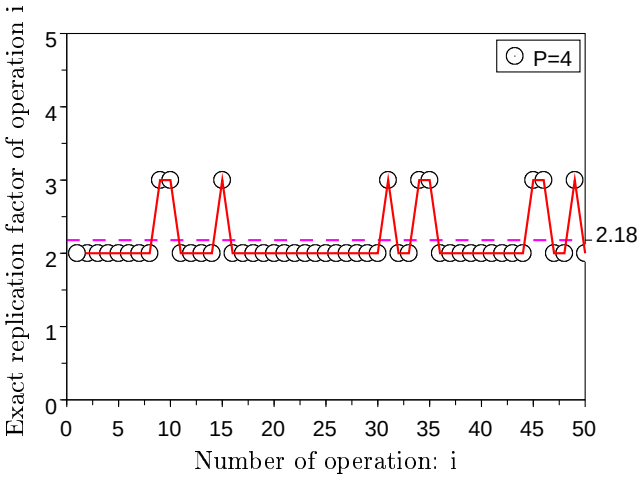

Figure 18: Exact replication factor of each operation for $\Lambda_{obj} = 10^{-5}$ on a 4 processors architecture.

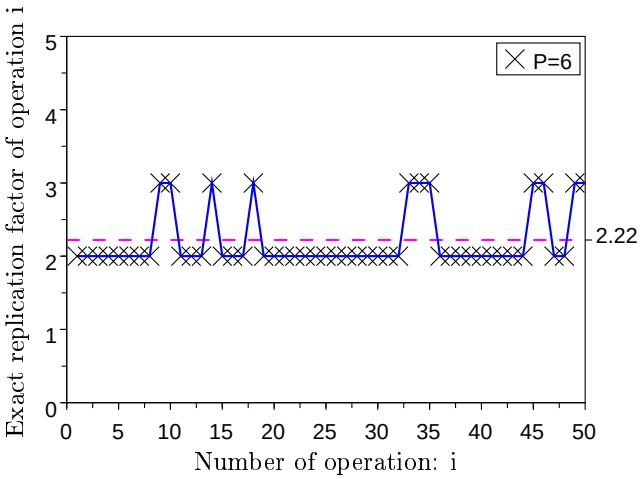

Figure 19: Exact replication factor of each operation for $\Lambda_{obj} = 10^{-5}$ on a 6 processors architecture.

6.5 Variation of the average replication factor in function of the processors' failure rate

We are also interested in the average replication factor of the operations scheduled (noted $AORP$) on each processor of the architecture. Recall that, for an operation X, $\Sigma(\mathrm{X})$ is the subset of processors that execute X. Hence, the replication factor of X is $|\Sigma(\mathrm{X})|$, and the number $AORP(\mathrm{P})$ is equal to:

$$AORP(\mathrm{P}) = \frac{\sum_{\mathrm{X\ s.t.}\ P \in \Sigma(\mathrm{X})} |\Sigma(\mathrm{X})|}{\sum_{\mathrm{X\ s.t.}\ P \in \Sigma(\mathrm{X})} 1} \tag{17}$$

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 intuitive because an operation scheduled onto a very reliable processor (i.e., whose failure rate is greater than Λ_{obj}) does not need to be replicated, while an operation scheduled onto an unreliable processor (i.e., whose failure rate is less than Λ_{obj}) must be replicated several times to reach Λ_{obj}.

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 actual

replication factor of each operation (see Section 5.1 and in particular Figure 13). For this reason, the simulation below (Figure 20) is run with three values of the failure rate of the communication media λ_m, different from those given in Tables 1 and 2: 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.

Figure 20 shows the $AORP$ of each processor P1 to P6 in the 6 processors architecture, 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}. This figure shows very clearly that $AORP$(P) is directly related to the failure rate per time unit of P, which demonstrates that our new BSH scheduling heuristic works remarkably well.

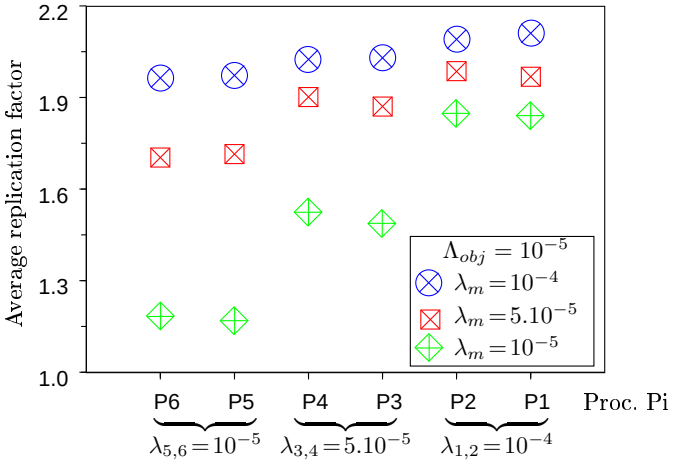

Figure 20: Average replication factor for each processor for $\Lambda_{obj} = 10^{-5}$.

As we can see in Figure 20, in the three cases, the more reliable the processor is, the less its average replication factor becomes: $AORP$(P5&P6) > $AORP$(P3&P4) > $AORP$(P1&P2). 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).

6.6 Variation of the schedule length overhead in function of Λ_{obj}

Figure 21 shows the length overhead of the schedules produced by BSH in function of the objective Λ_{obj}, averaged over the 50 $\mathcal{A}lg$ graphs, computed by equation (18):

$$\frac{length(BSH) - length(BSH \text{ without replication})}{length(BSH \text{ without replication})} \times 100 \qquad (18)$$

where "$length(BSH$ without replication)" denotes the average schedule length obtained by a modified BSH that does not replicate the tasks. This figure shows the compromise in terms of schedule length that the user has to pay in order to gain one or several orders of failure rate. Two curves are drawn, respectively for an architecture with 4 and 6 processors.

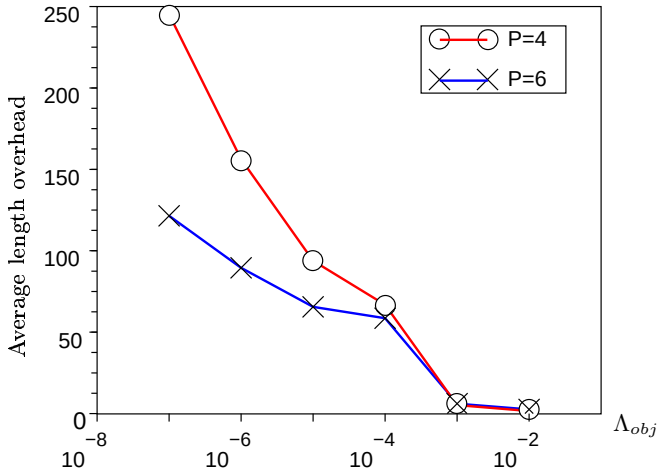

Figure 21: Variation of the length overhead in function of Λ_{obj}.

The overhead grows when Λ_{obj} decreases, because the replication factor increases and so does the number of replicas, hence the penalty incurred by the insertion of the routing operations becomes more and more important. Another reason for the overhead is that the insertion of routing operations (necessary to yield serial-parallel RBD) increases the schedule length. Finally, 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.

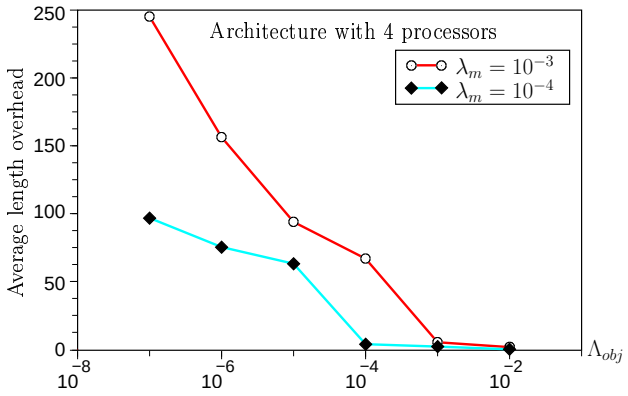

Figure 22: Variation of the schedule length overhead in function of λ_m.

Figure 22 gives the variations of the schedule length overhead in function of Λ_{obj}, for two different values of the failure rate per time unit of the communication links λ_m, in the 4 processors architecture. We can see that, 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}. Notice that the curve for $\lambda_m = 10^{-3}$ is identical to the curve for P= 4 in Figure 21.

6.7 Average schedule length overhead due to the routing operations in function of λ_m

Table 3 shows the average schedule length 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 our bicriteria scheduling algorithm 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 (19):

$$\frac{length(BSH) - length(BSH \text{ without routing})}{length(BSH \text{ without routing})} \times 100 \tag{19}$$

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 3 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}
P= 4	−4.12 %	+2.43 %	+4.09 %
P= 6	+2.44 %	+8.47 %	+9.96 %

Table 3: Average schedule length overhead due to the routing operations.

λ_m	10^{-3}	10^{-4}	10^{-5}
P= 4	2.07	1.50	1.33
P= 6	2.10	1.52	1.35

Table 4: 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 4); 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 communications and less schedule length overhead. Also, we can see that the overhead is greater for the 6 processors architecture than for the 4 processors one: this is because there is more concurrency available to schedule the communications in parallel in a fully connected 6 processors architecture (15 communication links) than in a 4 processors one (6 communication links): indeed, recall that the absence of routing operations means more data-dependencies in parallel (see Figure 7). Finally, the average overhead is only +3.88 %, which is very reasonable.

7 Conclusion

We have proposed a new framework for the (length,reliability) bicriteria static multiprocessor scheduling problem. Our first criteria remains the static schedule's length (crucial to assess the system's real-time property). For our second criteria, we consider the global failure rate (GSFR) of the system instead of the usual reliability. The GSFR is the failure rate of the multiprocessor system seen as if it were a single operation scheduled onto a single processor. The reason 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: first, 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; second, the fact that the length criteria overpowers the reliability criteria; and third, 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 from the other will always suffer for those three problems.

We have proposed a new bicriteria scheduling algorithm, called BSH, which takes as input a task DAG ($\mathcal{A}lg$), an 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}). It returns a static multiprocessor schedule of the task DAG onto the distributed architecture, such that its GSFR is less than Λ_{obj}. The GSFR is improved thanks to the active replication of the operations. To make the computation of the reliability of the partial schedules obtained during the execution of our BSH algorithm, 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 corresponding to the schedule is guaranteed to be serial-parallel, meaning that the reliability can always be computed in a linear time.

By invoking iteratively BSH with different values of Λ_{obj}, we are able to produce the Pareto curve for a given system (i.e., a given $\mathcal{A}lg$, $\mathcal{A}rc$, and $\mathcal{E}xe$), therefore providing the user with the choice among several tradeoffs 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 to 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).

Contents

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.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, January 2004.

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

[4] Y.G. Chen and M.C. Yuang. A cut-based method for terminal-pair reliability. *IEEE Trans. on Reliability*, 45(3):413–416, September 1996.

[5] W.W. Chu, M.-T. Lang, and J. Hellerstein. Estimation of inter-module communication (IMC) and its applications in distributed processing systems. *IEEE Trans. on Computers*, C-33:691–699, August 1984.

[6] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2/3):249–274, 2000.

[7] A. Dogan and F. Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. on 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] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] S. Kartik and C.S.R. Murthy. Task allocation algorithms for maximising reliability of distributed computing systems. *IEEE Trans. on Computers*, 46(6):719–724, June 1997.

[17] J.C. Knight and N.G. Leveson. An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Trans. on Software Engineering*, 12(1):96–109, 1986.

[18] B. Lisper. Trends in timing analysis. In *IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06*, pages 85–94, Braga, Portugal, October 2006. Springer.

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

[20] 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.

[21] P. Puschner and A. Burns. A review of worst-case execution-time analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.

[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. on Reliability*, 38(1):16–26, April 1989.

[24] S.M. Shatz, J.-P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. on Computers*, 41(9):1156–1168, September 1992.

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

[26] Y. Sorel. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation" methodology. In *Massively Parallel Computing Systems Conference*, pages 44–53, Ischia, Italy, May 1994.

[27] 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.

[28] S. Srinivasan and N.K. Jha. Safety and reliability driven task allocation in distributed systems. *IEEE Trans. on Parallel and Distributed Systems*, 10(3):238–251, March 1999.

[29] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separate cache and path analyses. *Real-Time Systems*, 18(2/3):157–179, May 2000.

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

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399