HAL
open science

Reliability and performance optimization of pipelined real-time systems

Anne Benoit, Fanny Dufossé, Alain Girault, Yves Robert

To cite this version:

Anne Benoit, Fanny Dufossé, Alain Girault, Yves Robert. Reliability and performance optimization of pipelined real-time systems. Journal of Parallel and Distributed Computing, 2013, 73 (6), pp.851-865. 10.1016/j.jpdc.2013.02.009 . hal-00926123

HAL Id: hal-00926123

https://inria.hal.science/hal-00926123v1

Submitted on 9 Jan 2014

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.

Reliability and performance optimization of pipelined real-time systems [1]

Anne Benoit, Fanny Dufossé, Alain Girault, and Yves Robert
ENS Lyon and INRIA Grenoble Rhône-Alpes, France
{Firstname.Lastname}@inria.fr

Abstract

We consider pipelined real-time systems that consist of a chain of tasks executing on a distributed platform. The processing of the tasks is pipelined: each processor executes only one interval of consecutive tasks. We are interested in minimizing both the input-output latency and the period of the application mapping. For dependability reasons, we are also interested in maximizing the reliability of the system. We therefore assign several processors to each interval of tasks, so as to increase the reliability of the system. Both processors and communication links are unreliable and subject to transient failures. We assume that the arrival of the failures follows a constant parameter Poisson law, and that the failures are statistically independent events. We study several variants of this multiprocessor mapping problem, with several hypotheses on the target platform (homogeneous/heterogeneous speeds and/or failure rates). We provide NP-hardness complexity results, and optimal mapping algorithms for polynomial problem instances. Efficient heuristics are presented to solve the general case, and experimental results are provided.

Keywords: Pipelined real-time systems, interval mapping, multi-criteria (reliability, latency, period) optimization, complexity results.

1. Introduction

A *pipelined real-time system* [22, 27] consists of a chain of tasks executing on a distributed platform. Each task is a block of code with a known amount of work to be processed. The role of the first task of the chain is to acquire some data set from the environment (thanks to sensor drivers), to process it, and finally to transmit its result to the second task. Each subsequent task receives its input data from its predecessor task, processes it, and transmits its result to its successor task, except the last task that transmits it to the environment (thanks to actuator drivers). The whole chain of tasks is executed repeatedly,

[1] Part of this work has appeared in ICPP'10.

as new data sets enter the system. Each data set is input to the first task and progresses from task to task until its processing is completed.

Executing a real-time system in a pipelined way is essential to increase the throughput, by making the best possible usage of available resources in the distributed execution platform. Tasks are assigned to processors using an *interval mapping*, which groups consecutive tasks of the linear chain and assigns them to the same processor. Interval mappings are more general than one-to-one mappings, which establish a unique correspondence between tasks and processors; they allow communication overheads to be reduced, not to mention the many situations where there are more tasks than processors, and where interval mappings are mandatory. The key performance-oriented metrics to determine the best interval mapping are the *period* and the *latency*. The period is the time interval between the beginning of the execution of two consecutive data sets. Equivalently, the inverse of the period is the *throughput*, which measures the aggregate rate of processing of data. The latency is the time elapsed between the beginning and the end of the execution of a given data set; hence, it measures the response time of the system for processing the data set entirely. Therefore, to minimize the period, we try to create many small intervals so that we can start processing the next data set as soon as possible, while for minimizing the latency, we rather try to reduce the sum of communication costs, and hence to split the chain of tasks in the least possible number of intervals. As a consequence, minimizing the latency is *antagonistic* to minimizing the period, and trade-offs should be found between these two criteria.

Each data set has a deadline on the completion time of its execution (the *real-time* constraint). The deadlines are related to the period P and latency L as follows. Data sets periodically enter the system with a given period P. Data set 0 enters the system at time 0 and has a deadline equal to L. Data set K enters the system at time $K \times P$ and has a deadline equal to $K \times P + L$. Accordingly, the deadline of each data set will be met as soon as we derive a schedule whose period does not exceed P, and whose latency does not exceed L. This model is consistent with those applications found in most safety critical real-time systems (e.g., avionics, railway or nuclear applications [8, 30]), which enforce a prescribed processing rate and maximum response time. This leads to a global deadline for each application instance (data set), but individual tasks distinct from the output task have no deadlines.

Besides constraints on the performance-oriented criteria, expressed as an upper bound on the period and/or the latency, pipelined real-time systems must also meet crucial *dependability constraints*, which are expressed as a lower bound on the *reliability* of the mapping. Increasing the reliability is achieved by replicating the intervals onto several processors. Augmenting the replication level (defined as the average number of times each interval is replicated) is good for reliability, but bad for period and latency, because fewer processors will be available for executing the task intervals. We thus have three antagonistic criteria: reliability, period, and latency. This antagonism between the criteria make the problem very challenging.

A typical example of applications with real-time and reliability constraints

is encountered in the automotive industry, with the Autosar architecture[2]. Autosar consists of a hardware architecture made of several processors (called ECUs – Electronic Computing Units) connected by a bus, and of several software components, each one being an embedded automotive function. Each function is a pipelined real-time system that starts with some input drivers that will generate a new dataset at each invocation (for instance the wheel angular speed), followed by several software blocks, and terminated by some actuator driver (for instance the hydraulic brake pressure). Each such function must meet a latency (also called the end-to-end timing constraint, from the sensor to the actuator), a period, and a reliability constraint.

We evaluate the reliability of a single task mapped onto a processor according to the classical model of Shatz and Wang [34], where each hardware component (processor or communication link) is *fail-silent* and is characterized by a *constant failure rate per time unit* λ: the reliability of a task of duration d is therefore $e^{-\lambda d}$. For an interval of several tasks mapped onto a single processor, we just have to sum up the task durations, hence obtaining $e^{-\lambda D}$, where D is the sum of task durations in the interval. For a mapping with replication, we compute the reliability by building the *Reliability Block Diagram* (RBD) [29, 3] corresponding to this mapping. Here we face the delicate issue that computing the reliability is exponential in the size of the mapping (or equivalently the size of the RBD). To solve this issue, we insert *routing operations* in the mapping to guarantee that the RBD is by construction serial-parallel, therefore allowing us to compute its reliability in linear time. The models are detailed in Section 2 and we discuss related work in Section 3.

Our contribution is multifold. In Section 4, we show how to compute the different objectives (reliability, expected and worst-case latency, expected and worst-case period) for a given multiprocessor mapping. Then, we derive complexity results for homogeneous platforms in Section 5. We prove that:

1. computing a mono-criterion mapping that optimizes the reliability is *polynomial* (Section 5.1);

2. optimizing both the reliability and the period remains *polynomial* (Section 5.2);

3. the problem of optimizing both the reliability and the latency is *NP-complete* (Section 5.3);

4. the problem of assigning processors for a given partition of the task chain into intervals is *polynomial* (Section 5.5).

Moreover, for homogeneous platforms, we provide a linear program to solve the problem of optimization of reliability for given bounds on period and latency in Section 5.4.

[2] AUTomotive Open System ARchitecture: `http://www.autosar.org`

For heterogeneous platforms, we prove that the mono-criterion problem of optimizing the reliability is *NP-complete*, and hence all the multi-criteria mapping problems that include the reliability in their criteria are also *NP-complete* (Section 6).

We provide heuristics in Section 7 for the most general problem of optimizing the reliability under constraints on period and latency on a heterogeneous platform, and we conduct experiments on homogeneous and heterogeneous platforms to assess their performance (Section 8). Finally, we state some concluding remarks and future research directions in Section 9.

2. Framework

In this section, we detail the application model, the platform model, the failure model, and the replication model. We end with the formal definition of the mono-criterion and multi-criteria multiprocessor mapping problems.

2.1. Application model

An application is a *chain* of n tasks $\mathcal{C} = (\tau_i)_{1 \leq i \leq n}$. Each task τ_i is a block of code that (1) receives its input from its predecessor τ_{i-1}, (2) computes a known amount of work, (3) and produces an output data set of a known size. Therefore, each task τ_i is represented by the pair (w_i, o_i), where w_i is the amount of work and o_i is the output data size. By convention, $o_n = 0$ because τ_n emits its result directly to the environment through actuator drivers. Specifying the size of the input data set required by a task is not necessary since, by definition of a chain, it is equal to the size of the output data set of its immediately preceding task. Figure 1 shows an example of a chain composed of n tasks.

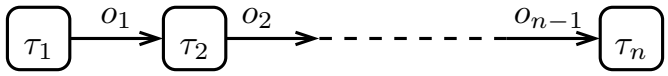

Figure 1: Example of a chain of n tasks.

Executing τ_i on a processor of speed s takes w_i/s units of time. Transmitting the result of τ_i on a link of bandwidth b takes o_i/b units of time. Knowing the values w_i and o_i is not a critical assumption since worst-case execution time (WCET) analysis has been applied with success to real-life processors actually used in embedded systems. In particular, it has been applied to the most critical existing embedded system, namely the Airbus A380 avionics software running on the Motorola MPC755 processor [15, 35].

2.2. Platform model

The target platform consists of p processors connected by point-to-point communication links. Let $\mathcal{P}$ be the processor set: $\mathcal{P} = (P_u)_{1 \leq u \leq p}$. We assume that communication links are *homogeneous*: this means that all links have the

same bandwidth b. On the contrary, each processor P_u may have a different speed s_u. Such platforms correspond to networks of workstations with plain TCP/IP interconnects or other LANs.

In order to derive a realistic communication model, we assume that the number of outgoing point-to-point connections of each processor is limited to $\mathcal{K}$. A given processor is thus capable of simultaneously sending messages to (and receiving messages from) $\mathcal{K}$ other processors. Indeed, there is no physical device capable of sending, say, 100 messages to 100 distinct processors, at the same speed as if it was a single message. The output bandwidth of the sender's network card would be a limiting factor. Our assumption of bounded multi-port communications [21] is reasonable for a large range of platforms, from large-scale clusters to multi-core System-on-Chips (SoCs).

In addition, we assume that communications are *overlapped* with computations, that is, a processor can compute the current instance of task τ_i and, in parallel, send to another processor the result of the previous instance of τ_i. This model is consistent with current processor architectures where a SoC can include a main processor and several communication co-processors.

2.3. Interval mapping

The chain of tasks is executed repeatedly in a *pipelined manner* to achieve a better throughput. As a consequence, mapping the chain on the platform involves dividing the chain into m intervals of consecutive tasks, and assigning each processor to a unique interval. This technique is known as *interval mapping*. Figure 2 shows an example of a division of a chain of tasks into m intervals.

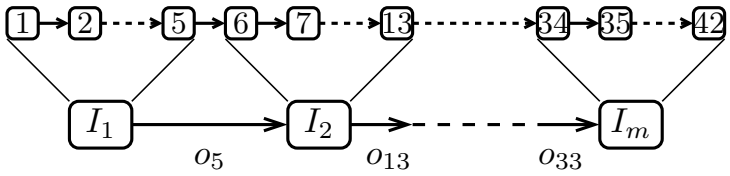

Figure 2: A chain of tasks divided into m intervals.

In a mapping without replication, each interval is assigned to a single processor. If the number of processors is greater than the number of tasks, then each interval can be of size one (that is, one task per interval), but this is rarely the case for real-life systems. Furthermore, having many small intervals is likely to decrease the period and the failure probability, but it will also increase the communication costs, and hence the latency; a trade-off must to be found.

In a mapping with replication, each interval is assigned to several processors. Replication is crucial to increase the reliability of the system [16], see Section 2.5 for details on the replication model.

For each $1 \leq j \leq m$, the interval I_j denotes the set of consecutive tasks between indices f_j and l_j. Moreover, $f_1 = 1$, $f_j = l_{j-1} + 1$ for $2 \leq j \leq m$, and

$l_m = n$. The amount of work processed by I_j is therefore $W_j = \sum_{\tau_i \in I_j} w_i = \sum_{i=f_j}^{l_j} w_i$. The size of the output data set produced by interval I_j is that of its last task, that is, o_{l_j}.

2.4. Failure model

Both processors and communication links can fail, and they are *fail-silent*. Classically, we adopt the failure model of Shatz and Wang [34]: failures are *transient* and the maximum 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 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 hardware components can have a failure rate around 10^{-6} per hour.

Since communication links are homogeneous, we note λ_ℓ their identical failure rate per time unit. Concerning the processors, we note λ_u the failure rate per time unit of processor P_u, for each P_u in $\mathcal{P}$. We assume that failure occurrences are *statistically independent events*. Note that transient failures are the most common failures in modern processors, all the more when processor voltage is lowered to reduce the energy consumption, because in that case, even very low energy particles are likely to create a critical charge leading to a transient failure [39].

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 $r = e^{-\lambda d}$, where λ is the failure rate per time unit of P or L. 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 task τ_i on processor P_u is:

$$r_{u,i} = e^{-\lambda_u \, w_i \, / \, s_u} \ . \tag{1}$$

Accordingly, the reliability of the interval I mapped on the processor P_u is:

$$r_{u,I} = e^{-\lambda_u \, W_j \, / \, s_u} = \prod_{\tau_i \in I} r_{u,i} \ . \tag{2}$$

Equations (1) and (2) show that platform heterogeneity may come from two factors: (i) processors having different speeds, and (ii) processors having different failure rates. We say that the platform is *homogeneous* if all the processors have the same speed s and the same failure rate λ (hence the reliability and the execution time of an interval no longer depend on the processor it is assigned to, and we use in that case the notation r_i instead of $r_{u,i}$ in Equation (1)); otherwise, we say that the platform is *heterogeneous*.

Finally, we let $r_{comm,i} = e^{-\lambda_\ell \, o_i \, / \, b}$ denote the reliability of the i-th communication.

2.5. Replication model

We use *spatial redundancy* to increase the reliability of a system: in other words, we replicate the intervals onto several processors. Figure 3 shows an example of mapping by interval with spatial redundancy: interval I_1 is mapped on processors $\{P_1, P_2, P_3\}$, interval I_2 is mapped on processors $\{P_4, P_5\}$, and so on until interval I_m, which is mapped on processors $\{P_{p-1}, P_p\}$. Concerning communications, the data-dependency o_{l_1} is mapped on the point-to-point links $\{L_{14}, L_{15}, L_{24}, L_{25}, L_{34}, L_{35}\}$, and so on (where L_{uv} denotes the link between P_u and P_v).

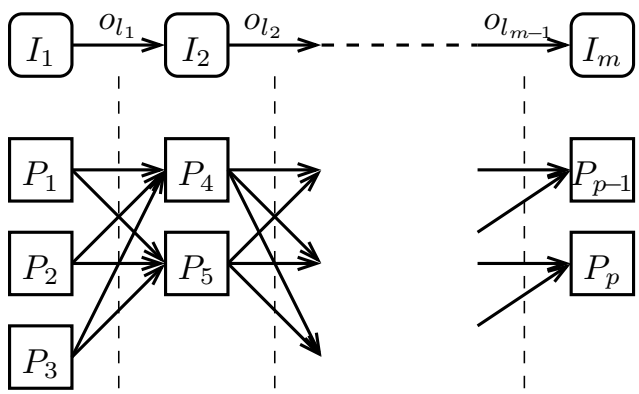

Figure 3: An example of interval mapping.

To increase the reliability, *each* processor of a given interval communicates with *each* processor of the next interval. Specifically, for any $1 \leq j \leq m-1$, all the processors executing interval I_j send their result to all the processors executing the next interval I_{j+1}. Because of the bounded number $\mathcal{K}$ of possible communications (see Section 2.2), the maximum number of replicas per interval is also limited to $\mathcal{K}$.

2.6. Multiprocessor mapping problem

We study several variants of the multiprocessor interval mapping problem. The inputs of the problem are a chain of n tasks $\mathcal{C} = (\tau_i)_{1 \leq i \leq n}$, a hardware platform of p processors $\mathcal{P} = (P_u)_{1 \leq u \leq p}$, and a bound $\mathcal{K}$ on the maximum number of replications for each interval of tasks. The output is an interval mapping of $\mathcal{C}$ onto $\mathcal{P}$, that is, a distribution of $\mathcal{C}$ into m intervals and an assignment of each interval to at most $\mathcal{K}$ processors of $\mathcal{P}$, such that each processor executes only one interval. Each variant of the mapping problem optimizes a different set of criteria among the following:

- the reliability,
- the expected input-output latency,

- the worst-case input-output latency,
- the expected period,
- the worst-case period.

3. Related work

Several papers have dealt with workflow applications whose dependence graph is a linear chain. The pioneering papers [36, 37] investigate bi-criteria (period, latency) optimization of such workflows on homogeneous platforms. An extension of these results to heterogeneous platforms is provided in [6, 7]. Yet, all these papers assume the platform to be reliable.

In our previous work [5], we studied the (reliability, latency) mapping problem with fail-silent processors. The model in [5] is quite different, and much more crude, than the one of this paper: each processor has an absolute probability of failing, independent of task durations, and the faults are unrecoverable.

To the best of our knowledge, we are not aware of other published work on optimizing workflows for reliability focusing on the particular case of pipelined chain workflows. However, many papers have dealt with a directed acyclic graph (DAG) instead of a pipelined workflow, be it a fully general DAG [13], a linear chain [32], or even independent tasks [23, 32]. The closest paper to our present work is [32]: it contains a short section on linear chains, with a mono-criterion dynamic programming algorithm for optimizing the reliability, which is similar to Algorithm 1 (see Section 5.1). Other recent papers have also addressed the problem of fault-tolerant mapping of streaming applications on multiprocessors [33], of lifetime-aware mapping on multiprocessors [20] (lifetime-aware meaning taking into account the temperature, the supply voltage, the current density, and the application and architecture characteristics), and remapping strategies for fault-tolerance [11, 26]. However, all these papers adopt models (regarding the application, the architecture, or the failures) that differ from our own models.

Linear chains of tasks are a particular case of DAGs, so existing algorithms for general DAGs can be used to solve our some of our mono-criterion or bi-criteria optimization problems. However, no result is available for the tri-criteria (period, latency, reliability) problems. In addition, on the theoretical side, our simpler application framework allows us to obtain deeper results (NP-completeness and optimal algorithms in the homogeneous case); and on the practical side, it allows us to derive more efficient scheduling algorithms (dynamic programming heuristics in the heterogeneous case).

The specific problem of bi-criteria (length, reliability) multiprocessor scheduling has also been addressed in [12, 1, 19, 31, 17, 18] for general DAGs of operations, but except [1, 17, 18], these papers do not replicate the operations and have thus a very limited impact on the reliability. Moreover, none of them consider chains of tasks and interval mappings, and therefore they attempt to minimize the length of the mapping without distinguishing between the period and the latency (the latter one being equivalent to the schedule length).

More generally, our application model is quite similar to that presented in [22]. A formal definition of pipelined real-time systems can be found in [38]. Another pipelined system model is presented in [27, 28], where several chains of periodic tasks are computed in parallel, and there is a deadline for each task instance. Real-time systems constituted of chains of tasks with end-to-end deadlines have also been studied in [9].

4. Evaluation of a given mapping

In this section, we detail the computation of the different objectives (reliability, expected and worst-case latency, expected and worst-case period) for a given mapping. We compute the reliability of a mapping by building its *reliability block diagram* (RBD) [29, 3]. Formally, a RBD is an *acyclic oriented graph* (N, E), where each node of N is a *block* representing an element of the system, and each arc of E is a *causality link* between two blocks. Two particular connection points are its *source* S and its *destination* D. A RBD is *operational* if and only if there exists at least one operational path from S to D. A path is operational if and only if all the blocks in this path are operational. The probability that a block is operational is its reliability. By construction, the probability that a RBD is operational is equal to the reliability of the system that it represents.

In our case, the system is the multiprocessor interval mapping, possibly partial, of the application on the platform. A mapping is *partial* if not all intervals have been mapped yet, but of course those intervals that are mapped are such that all their predecessors are also mapped. Each block of the RBD represents an interval I_j placed on a processor or a data-dependency o_{l_j} between the two intervals I_j and I_{j+1} placed on a communication link. The reliability of a block is therefore computed according to Equation (2).

Computing the reliability in this way assumes that the occurrences of the failures are statistically independent events (see Section 2.4). 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 necessarily be the case for software faults [25].

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. The reason is that processors are heterogeneous: the completion dates of a given interval on its assigned processors are different, so the reception dates by the processors of the next interval are different. This is true even when the application is a chain of intervals rather than a general graph. See Figure 4 for an illustration, where the RBD corresponding to the mapping has no specific form.

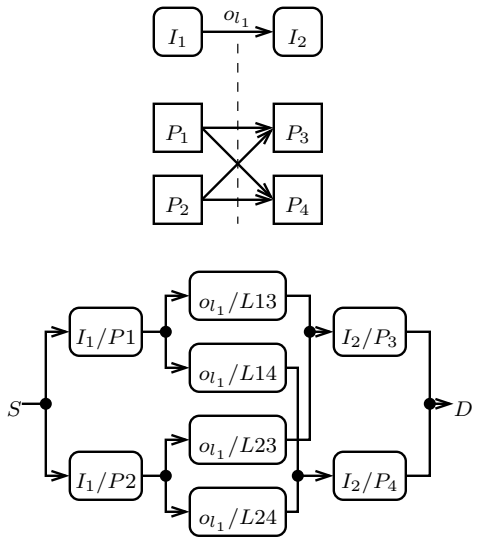

Figure 4: A mapping of two intervals (I_1 and I_2) on four processors (P_1 to P_4) and its RBD which has no particular form.

One solution for computing the reliability of the mapping of Figure 4 involves enumerating all the *minimal cut sets* of its RBD [24]. A *cut set* in a RBD is a set of blocks C such that there is no path from S to D if all the blocks of C are removed from the RBD. A cut C is *minimal* if, whatever the block that is removed from it, the resulting set is not a cut anymore. It follows that the reliability of a minimal cut set is the reliability of all its blocks put in parallel. The reliability of the mapping can then be approximated by the reliability of the alternative RBD composed of all the minimal cut sets put in sequence. Because this RBD is *serial-parallel*, this computation is linear in the number of minimal cut sets. The problem is that, in general, the number of minimal cuts is exponential in the size of the RBD [24].

For this reason, we follow the approach of [17] and we insert *routing operations* between the intervals to make sure that the RBD representing a mapping is always *serial-parallel*, therefore making tractable the computation of the reliability. This is illustrated in Figure 5, where a routing operation R has been mapped on processor P_5 and the RBD corresponding to the mapping is serial-parallel; as a consequence, the reliability of this mapping can be computed in a linear time w.r.t. the number of intervals.

Routing operations can be mapped on *any* processor. For instance, in the RBD of Figure 5, R could have been mapped on P_1 instead of P_5, therefore avoiding the need for the communication (o_{l_1}/L_{15}). Also, routing operations are assumed to be executed in 0 time units [17]. As a consequence, for any processor P_u, the reliability of the block (R/P_u) is 1.

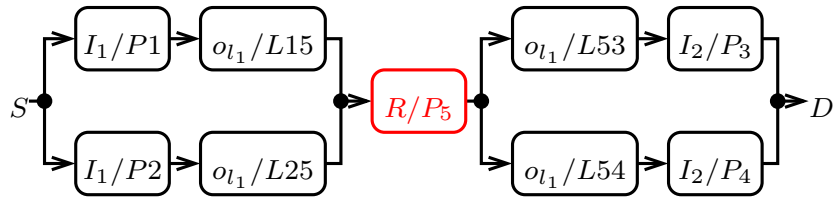

Figure 5: The serial-parallel RBD obtained from the same mapping as in Figure 4 but with an additional routing operation R.

As we have advocated, inserting routing operations yields the huge advantage of making the reliability computation linear in time. This comes at a cost in the execution time of the system because of the increased number of communications. For instance, in Figure 5, o_{l_1} is transmitted twice before reaching I_2. However, it has been shown in [17] that the overhead incurred by the routing operations is reasonable (only +3.88 % on average).

For an interval I of weight W mapped on the subset of processors $\mathcal{P}_I$, let ec be its expected time of computation, and let wc be its worst-case execution time (by the slowest processor of $\mathcal{P}_I$). Assume that the processors in $\mathcal{P}_I$ are ordered according to their speed, from the fastest P_1 to the slowest P_t: that is, $\forall 1 \leq u < t$, we have $s_u \geq s_{u+1}$. Then, the expected and worst-case execution times of I on $\mathcal{P}_I$ are expressed in Equations (3) and (4). Equation (3) sums up, for each P_u, the case where the first $u-1$ fastest processors fail, and the u-th one is successful.

$$ec(I, \mathcal{P}_I) = W \times \frac{\sum_{u=1}^{t} \left(\frac{1}{s_u} r_{u,I} \prod_{v=1}^{u-1} (1 - r_{v,I}) \right)}{1 - \prod_{u=1}^{t} (1 - r_{u,I})} \; ; \tag{3}$$

$$wc(I, \mathcal{P}_I) = \frac{W}{s_t} \; . \tag{4}$$

Then, for a mapping $(I_1, \mathcal{P}_1), \ldots, (I_m, \mathcal{P}_m)$, the expected latency EL and the expected period EP are:

$$EL = \sum_{i=1}^{m} ec(I_i, \mathcal{P}_i) + \frac{o_i}{b} \; ; \tag{5}$$

$$EP = \max \left\{ \max_{1 \leq i \leq m} \left\{ \frac{o_i}{b} \right\}, \max_{1 \leq i \leq m} ec\left(I_i, \mathcal{P}_i\right) \right\} \; . \tag{6}$$

The worst-case latency WL and the worst-case period WP are defined similarly, but with the worst-case cost of intervals (Equation (4)) instead of the

expected cost (Equation (3)):

$$WL = \sum_{i=1}^{m} wc(I_i, \mathcal{P}_i) + \frac{o_i}{b} \; ; \tag{7}$$

$$WP = \max \left\{ \max_{1 \le i \le m} \left\{ \frac{o_i}{b} \right\}, \max_{1 \le i \le m} wc\left(I_i, \mathcal{P}_i\right) \right\} \; . \tag{8}$$

Finally, thanks to the routing operations, the reliability of the mapping $(I_1, \mathcal{P}_1), \ldots, (I_m, \mathcal{P}_m)$ is:

$$r = \prod_{i=1}^{t} \left(1 - \prod_{P_u \in \mathcal{P}_i} (1 - r_{comm,i-1} \times r_{u,I_i} \times r_{comm,i}) \right) \; . \tag{9}$$

Equation (9) above is computed according to the generic form of the RBD of Figure 5. To account for the fact that the first interval I_1 has no incoming communication, we just set $o_0 = 0$, hence $r_{comm,0} = 1$. The same occurs for the outgoing communication of the last interval I_m. Finally, routing operations do not appear in Equation (9) since their reliability is always equal to 1.

5. Complexity results for homogeneous platforms

In this section, we provide optimal polynomial-time algorithms for the mono-criterion reliability optimization problem, and then for the bi-criteria (reliability, period) optimization problem. Then, we prove the NP-completeness of the bi-criteria (reliability, latency) optimization problem. We provide an integer linear program to solve the tri-criteria problem and a polynomial-time algorithm to optimally allocate processors for a given partition of the chain of tasks in intervals. Note that on homogeneous platforms, the expected latency and worst-case latency are the same. This also holds true for the expected period and worst-case period.

5.1. Reliability optimization

We present a mono-criterion polynomial-time algorithm that maximizes the reliability of a given chain of tasks on a given homogeneous platform. Algorithm 1 is a dynamic programming algorithm. It is a simplified version of Algorithm 2 for bi-criteria (reliability, period) optimization, which we present in the next section.

Theorem 1. *Algorithm 1 computes in time $O(n^2p^2)$ the optimal mapping for reliability optimization on fully homogeneous platforms.*

Proof. In this algorithm, $F(i, k)$ is the optimal reliability when mapping the first i tasks on k processors, and it is computed iteratively with the dynamic programming procedure. The recursive formula can be found on line 10 of Algorithm 1. For any number of tasks of the last interval, and for any number of replications of this last interval, the formula computes the optimal reliability for previous tasks and available processors, and it selects the solution with optimal reliability. □

Algorithm 1: Optimal algorithm for reliability optimization on fully homogeneous platforms.

Data: a number p of fully homogeneous processors of failure rate λ, a chain A of n tasks of sizes w_i, and a maximum number $\mathcal{K}$ of replications

Result: a reliability r

for $k = 1$ **to** $\min\{\mathcal{K}, p\}$ **do**

$$F(1,k) = 1 - \left(1 - r_{comm,0} \times r_1 \times r_{comm,1}\right)^k \ ;$$

end

$F(0,0) = 1$;

for $i = 1$ **to** n **do**

  $F(i,0) = 0$;

end

for $i = 2$ **to** n **do**

  for $k = i$ **to** p **do**

$$F(i,k) = \max_{1 \le j < i, 1 \le q \le \min\{\mathcal{K},k\}} \Bigg\{ F(j, k-q) \times \left(1 - \left(1 - r_{comm,j-1} \times \prod_{j \le l \le i} r_l \times r_{comm,i}\right)^q\right) \Bigg\} \ ;$$

  end

end

$r = \max_{1 \le q \le p} F(n,q)$;

5.2. Reliability/period optimization

We now present a bi-criteria (reliability, period) polynomial-time algorithm that optimizes the reliability of a mapping given a bound on the period. Recall that, for homogeneous platforms, the worst-case period and the expected period are the same.

Algorithm 2: Optimal algorithm for reliability optimization on fully homogeneous platforms, when a bound on the period is given.

Data: a number p of fully homogeneous processors of failure rate λ, a chain A of n tasks of sizes w_i, a maximum number $\mathcal{K}$ of replications, and an upper bound P on the period

Result: a reliability r

```
for k = 1 to min{K, p} do
    if max(o_0/b, w_1/s, o_1/b) ≤ P then
        F(1,k) = (1 − (1 − r_{comm,0} × r_1 × r_{comm,1})^k) ;
    else
        F(1,k) = 0;
    end
end
for i = 1 to n do
    F(i,0) = 0;
end
for i=2 to n do
    for k=i to p do
        F(i,k) = max_{1≤j<i, 1≤q≤min{K,k}} { F(j,k−q) ×
            (1 − (1 − r_{comm,j} × ∏_{j<l≤i} r_l × r_{comm,i})^q)
            | max(o_j/b, (Σ_{v=j+1}^{i} w_v)/s, o_i/b) ≤ P } ;
    end
end
r = max_{1≤q≤p} F(n,q);
```

Theorem 2. *Algorithm 2 computes in time $O(n^2p^2)$ the optimal mapping for reliability optimization on fully homogeneous platforms, when a bound on the period is given.*

Proof. In this algorithm, $F(i,k)$ is again the optimal reliability when mapping the first i tasks on k processors. The dynamic programming procedure of Algorithm 1 has been modified to account for the period bound. The algorithm only considers intervals enforcing the period bound. □

Finally, we observe that the converse problem, namely optimizing the period when a bound on the reliability is enforced, is polynomial too. We can simply perform a binary search on the period and repeatedly execute Algorithm 2 until the optimal value is found.

5.3. Reliability/latency optimization

We now prove the NP-completeness of the bi-criteria (reliability, latency) optimization problem on homogeneous platforms. As for the period, there is no difference between the worst-case latency and the expected latency on such platforms.

Theorem 3. *The problem of optimizing the reliability on homogeneous platforms, with a bound on the latency, is NP-complete.*

Proof. Consider the associated decision problem: given a homogeneous platform, a chain of tasks, a bound $\mathcal{K}$ on the number of replications, a reliability r, and a latency L, does there exist a mapping whose reliability is at least r and whose latency is at most L? This problem is obviously in NP: given a mapping, it is easy to compute its reliability and latency, and to check that it is valid in polynomial time.

To establish the completeness, we use a reduction from 2-PARTITION (instance $\mathcal{I}_1$): given a set A of n numbers $a_1, \ldots, a_n$, does there exist a subset $A' \subset A$ such that $\sum_{a \in A'} a = \sum_{a \notin A'} a$. Let $T = \frac{1}{2} \sum_{a \in A} a$. Let $a_{min} = \min_{1 \le i \le n}\{a_i\}$ and $a_{max} = \max_{1 \le i \le n}\{a_i\}$. We build the following instance $\mathcal{I}_2$ of our problem with $3n+1$ tasks and $6n$ identical processors:

- $\mathcal{K} = 2$ and $\lambda = 10^{-8} 10^{-n} a_{max}^{-3n}$;
- $s = b = 1$ (unit processor speed and link bandwidth);
- $B = \frac{1}{2a_{min}} \left(\frac{n}{4} + n a_{max}^2 + T + 2\right)$;
- $\forall 1 \le i \le n$, $w_{3i-2} = B$, $w_{3i-1} = \frac{1}{2}$ and $w_{3i} = a_i$;
- $w_{3n+1} = B$;
- $\forall 1 \le i \le n, r_i = e^{-\lambda w_i}$ and $r_{comm,i} = 1$;
- $\forall 1 \le i \le n$, $o_{3i-2} = 0$, $o_{3i-1} = a_i$ and $o_{3i} = 0$;
- $L = (n+1)B + \frac{n}{2} + 3T$.

It follows that the reliability of the mapping is:

$$r = \left(1 - (1 - e^{-\lambda B})^2\right)^{n+1} \times \left(1 - \lambda^2 (\tfrac{n}{4} + \textstyle\sum_{1 \le i \le n} a_i^2 + T) - \lambda^4 \times 2^{2n} (a_{max} + 1)^n\right)$$

The size of instance $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution. Suppose first that $\mathcal{I}_1$ has a solution A'. Then we propose the following solution for $\mathcal{I}_2$:

- all intervals are replicated 2 times;
- any task of size B make up an interval;
- for all $1 \leq i \leq n$, if $a_i \in A'$, then T_{3i-1} and T_{3i} are assigned to two different intervals, else they constitute one single interval.

This yields the following costs for the latency:

- the sum of computation costs does not depend of the mapping: $(n+1)B + \frac{n}{2} + 2T$;
- for each $a_i \in A'$, we add a communication cost a_i.

We thus obtain a latency $L = (n+1)B + \frac{n}{2} + 3T$. Concerning the reliability, it is the product of the reliability of all intervals:

- the reliability of intervals of size B is $(1-(1-e^{-\lambda B})^2)$;
- for each $a_i \in A'$, the product of the reliability of the two intervals for tasks T_{3i-1} and T_{3i-1} is $(1-(1-e^{-\frac{\lambda}{2}})^2)(1-(1-e^{-\lambda a_i})^2)$, which is greater than $(1-\frac{\lambda^2}{4})(1-\lambda^2 a_i^2)$;
- for each $a_i \notin A'$, the reliability of the interval for tasks T_{3i-1} and T_{3i-1} is $(1-(1-e^{-\lambda(a_i+\frac{1}{2})})^2)$, which is greater than $1-\lambda^2(a_i+\frac{1}{2})^2$.

We thus obtain, for the product of all these reliabilities,

$$\begin{aligned}
r' \;=\; & (1-(1-e^{-\lambda B})^2)^n \times \\
& \textstyle\prod_{a_i \in A'} (1-(1-e^{-\frac{\lambda}{2}})^2)(1-(1-e^{-\lambda a_i})^2) \times \\
& \textstyle\prod_{a-i \notin A'} \left(1-\left(1-e^{-\lambda\left(a_i+\frac{1}{2}\right)}\right)^2\right) \\
\geq\; & (1-(1-e^{-\lambda B})^2)^n \times \\
& \textstyle\prod_{a_i \in A'} (1-\frac{\lambda^2}{4})(1-\lambda^2 a_i^2) \times \\
& \textstyle\prod_{a-i \notin A'} (1-\lambda^2(a_i+\frac{1}{2})^2) \\
\geq\; & (1-(1-e^{-\lambda B})^2)^n \times \\
& \left(1-\lambda^2\left(\frac{n}{4}+\textstyle\sum_{1\leq i\leq n} a_i^2+T\right)-\lambda^4 2^{2n}(a_{max}+1)^n\right)
\end{aligned}$$

Suppose now that $\mathcal{I}_2$ has a solution. The exponent in the reliability bound implies that any interval is replicated at least 2 times, and the bound on replication is 2. This means that all intervals are replicated exactly 2 times. Suppose that one of the tasks of size B is computed together with another task in the

same interval. This yields the bound on reliability:

$$\begin{aligned}
r' &< (1-(1-e^{-\lambda B})^2)^n(1-(1-e^{-\lambda(B+a_{min})})^2) \\
&< (1-(1-e^{-\lambda B})^2)^{n+1} \times \tfrac{1-\lambda^2(B+a_{min})^2}{1-\lambda^2 B^2(1-\frac{\lambda B}{2})^2} \\
&< (1-(1-e^{-\lambda B})^2)^{n+1}(1-\lambda^2(B+a_{min})^2) \\
&\quad (1+\lambda^2 B^2(1-\tfrac{\lambda B}{2})^2 + 2\lambda^4 B^4(1-\tfrac{\lambda B}{2})^4) \\
&< (1-(1-e^{-\lambda B})^2)^{n+1} \times (1-2\lambda^2 B a_{min} + 7\lambda^4 B^4) \\
&< r
\end{aligned}$$

This means that any task of size B makes up an interval. Let A' be the set of values i such that T_{3i-1} and T_{3i} are not in the same interval. We obtain the following formulas:

- For the reliability:

$$\begin{aligned}
r \leq\ & (1-(1-e^{-\lambda B})^2)^n \times \\
& \textstyle\prod_{a_i \in A'}(1-(1-e^{-\frac{\lambda}{2}})^2)(1-(1-e^{-\lambda a_i})^2) \times \\
& \textstyle\prod_{a_i \notin A'}\left(1-\left(1-e^{-\lambda\left(a_i+\frac{1}{2}\right)}\right)^2\right) \\
\leq\ & (1-(1-e^{-\lambda B})^2)^n \times \\
& \textstyle\prod_{a_i \in A'}(1-\frac{\lambda^2}{4}(1-\frac{\lambda}{4})^2)(1-\lambda^2 a_i^2(1-\lambda a_i)^2 \times \\
& \textstyle\prod_{a_i \notin A'}(1-(\lambda^2+\frac{\lambda^2}{4}+\lambda^2 a_i)(1-\frac{\lambda}{2}(a_i+\frac{1}{2}))^2) \\
\leq\ & 1-\lambda^2(\tfrac{n}{4}+\textstyle\sum_{1\leq i\leq n} a_i^2 + \sum_{a_i \notin A'} a_i) + \lambda^3 10^n a_{max}^{3n}
\end{aligned}$$

- For the latency:

$$(n+1)B + \frac{n}{2} + \sum_{a_i \in A'} a_i + 2T \leq (n+1)B + \frac{n}{2} + 3T$$

This means $\sum_{a_i \notin A'} a_i \leq T$ and $\sum_{a_i \in A'} a_i \leq T$. Hence, A' is a solution for $\mathcal{I}_1$. This concludes the proof. □

We conclude that, on homogeneous platforms, the bi-criteria (reliability, period) problem is polynomial, while the bi-criteria (reliability, latency) problem is NP-complete. As a consequence, the tri-criteria (reliability, period, latency) problem is NP-complete too.

It is striking, and somewhat unexpected, that the bi-criteria (reliability, period) problem is easier than the (reliability, latency) problem. The intuition for this difference is the following: when the period bound is given, we know once and for all which processors are fast enough to be enrolled for a given interval. Therefore, mapping decisions are local. On the contrary, the computation of the latency remains global, and its final value, including communication costs, depends upon decisions that will be made further on.

5.4. Integer linear program

In this section, we show how to derive an integer linear program (ILP) to solve the following problem: given an instance with n tasks and p homogeneous processors, bounds P on period and L on latency, compute the most reliable schedule respecting both bounds. Despite its high computation complexity, this ILP will be used on small problem instances to assess the absolute performance of the heuristics (see Section 8).

The ILP has $O(n^2 \times p)$ variables: for $1 \leq i \leq j \leq n$ and $1 \leq k \leq \min(p, \mathcal{K})$, $a_{i,j,k} = 1$ if the interval $\tau_i, ..., \tau_j$ is allocated onto k processors, and $a_{i,j,k} = 0$ otherwise. The objective function is the logarithm R of the reliability, which we want to maximize.

We list below the constraints that need to be enforced.

- Each task τ_i is included in exactly one interval:

$$\forall 1 \leq i \leq n, \sum_{1 \leq j \leq i} \sum_{i \leq k \leq n} \sum_{1 \leq \ell \leq p} a_{i,j,\ell} = 1 \ .$$

- At most p processors are used:

$$\sum_{1 \leq i \leq j \leq n} \sum_{1 \leq k \leq \mathcal{K}} k \times a_{i,j,k} \leq p \ .$$

- The latency bound is enforced:

$$\sum_{1 \leq i \leq j \leq n} \sum_{1 \leq k \leq \mathcal{K}} \frac{1}{s} \left(\sum_{i \leq \ell \leq j} w_\ell \right) \times a_{i,j,k} \leq L \ .$$

- The period bound is enforced:

$$\forall 1 \leq i, j \leq n, \frac{1}{s} \left(\sum_{i \leq \ell \leq j} w_\ell \right) \times \sum_{1 \leq k \leq \mathcal{K}} a_{i,j,k} \leq P \ ;$$

$$\forall 1 \leq i, j \leq n, o_j \times \sum_{1 \leq k \leq \mathcal{K}} a_{i,j,k} \leq P \ .$$

Finally, the objective function is to maximize the logarithm of the reliability:

$$\max \sum_{1 \leq i \leq j \leq n} \sum_{k=1}^{\mathcal{K}} \log \left(1 - (1 - \exp^{\frac{\lambda}{s} \sum_{l=i}^{j} w_l})^k \right) \times a_{i,j,k} \ .$$

5.5. Allocation of intervals to processors

In this section, we consider that the partition into intervals is given, and we search for the best allocation of these intervals across the processors. This sub-problem is used in particular while designing heuristics in Section 7.

Once the intervals are fixed, since the platform is homogeneous, the period and latency are fixed. The allocation of processors only impacts the reliability. We derive below an optimal algorithm, Algo-Alloc, which assigns processors to intervals in order to maximize the reliability. The main idea is to allocate processors one by one to intervals, and the next interval is greedily chosen so as to maximize the current reliability. The algorithm Algo-Alloc is the following:

- initially, we allocate one processor onto each interval;
- then, while there remains an un-allocated processor and an interval replicated less than $\mathcal{K}$ times, we allocate a new processor on the interval whose ratio
$$\frac{\text{reliability with one more replica processor}}{\text{current reliability}}$$
is maximum.

We prove below the optimality of this greedy algorithm:

Theorem 4. *Given a partition into intervals, Algorithm* Algo-Alloc *maximizes the reliability of the allocation.*

Proof. Let $I_1 \to \cdots \to I_i$ be the chain of intervals, where W_j (resp. o_j) is the computation cost (resp. communication cost) of interval I_j, for $1 \leq j \leq i$. Moreover, let k_j be the number of processors allocated to interval I_j by Algorithm Algo-Alloc, and let k'_j be the number of such processors in an optimal solution.

First, note that if $i \times \mathcal{K} \leq p$, Algo-Alloc allocates $\mathcal{K}$ processors per interval, and this is optimal, i.e., $\forall 1 \leq j \leq i$, $k_j = k'_j = \mathcal{K}$. Otherwise, all processors are allocated in both solutions ($\sum_{j=1}^{i} k_j = \sum_{j=1}^{i} k'_j = p$), because replicating an interval one more time always increases reliability. To see this, for $2 \leq k \leq \mathcal{K}$, consider the increase in reliability when assigning a k-th processor to interval I_j:

$$R_{k,j} = \frac{\text{reliability of } I_j \text{ with } k \text{ processors}}{\text{reliability of } I_j \text{ with } k-1 \text{ processors}} .$$

We obtain $R_{k,j} = \frac{1-\alpha_j^k}{1-\alpha_j^{k-1}}$, with $\alpha_j = 1 - \exp^{-\lambda \frac{W_j}{s}}$. Note that this value is independent of the allocation of the other intervals. We derive:

$$\begin{aligned} R_{k+1,j} - R_{k,j} &= \frac{(1-\alpha_j^{k+1})(1-\alpha_j^{k-1}) - (1-\alpha_j^k)^2}{(1-\alpha_j^k)(1-\alpha_j^{k-1})} \\ &= \frac{2\alpha_j^k - \alpha_j^{k+1} - \alpha_j^{k-1}}{(1-\alpha_j^k)(1-\alpha_j^{k-1})} \\ &\leq 0 . \end{aligned}$$

by convexity of the function $x \to \alpha^x$. The ratio $R_{k,j}$ is thus decreasing with k.

Now suppose that there exist two values j_1 and j_2 with $k_{j_1} < k'_{j_1}$ and $k_{j_2} > k'_{j_2}$. To simplify notations, assume that $j_1 = 1$ and $j_2 = 2$, hence $k_1 < k'_1$ and $k_2 > k'_2$. Consider the iteration of Algo-Alloc during which the (k'_2+1)-th processor is added to interval I_2. At that point, there were $k^* \leq k_1$ processors assigned to I_1. By construction of Algo-Alloc, since interval I_2 is chosen, we have $R_{k'_2+1,2} \geq R_{k^*+1,1}$. Also, $R_{k^*+1,1} \geq R_{k'_1,1}$ because $k^* \leq k_1 < k'_1$ and $R_{k,1}$ is decreasing with k. We thus have $\frac{R_{k'_2+1,2}}{R_{k'_1,1}} \geq 1$, but this latter quantity is the variation of the global reliability when reassigning one processor from I_1 to I_2. Hence the allocation $(k'_1-1, k'_2+1, k'_3, ..., k'_i)$ is at least as reliable as the original optimal allocation $(k'_1, k'_2, k'_3, ..., k'_i)$, and therefore they are both optimal.

After a finite number of such reassignments, we obtain the allocation of Algo-Alloc, thereby establishing its optimality. □

6. Complexity results for heterogeneous platforms

In this section, we prove the NP-completeness of the mono-criterion reliability optimization problem on heterogeneous platforms.

Theorem 5. *The problem of optimizing the reliability on heterogeneous platforms is NP-complete.*

Proof. Consider the associated decision problem: given a heterogeneous platform, a chain of tasks, a bound on the number $\mathcal{K}$ of replications, and a reliability r, does there exist a mapping of reliability at least r? This problem is obviously in NP: given a reliability and a mapping, it is easy to compute the reliability and to check that it is valid in polynomial time.

To establish the completeness, we use a reduction from 3-PARTITION. Consider the following general instance $\mathcal{I}_1$ of 3-PARTITION: given $3n$ numbers $a_1, \ldots, a_{3n}$ and a number T such that $\sum_{1 \leq j \leq 3n} a_j = nT$, does there exist n independent subsets $B_1, \ldots, B_n$ of $\{a_1, \ldots, a_{3n}\}$ such that for all $1 \leq i \leq n$, $\sum_{a_j \in B_i} a_j = T$?

We build the following instance $\mathcal{I}_2$ with n tasks and $p = 3n$ processors:

- $\lambda = \frac{10^{-8}}{nT^2}$;
- $\mathcal{K} = 3$;
- $\gamma = 1 + \frac{1}{2(T-1)}$;
- $\forall 1 \leq i \leq n, w_i = 1/n$ (all tasks have cost $1/n$);
- $r_{u,i} = e^{-\lambda_u \frac{w_i}{s_u}}$;
- $r_{comm,i} = 1$;
- $\forall 1 \leq u \leq 3n, \lambda_u = \lambda * \gamma^{a_u}$ and $s_u = 1$.

It follows that the reliability r of the mapping is $r = \left(1 - \lambda^3\gamma^T\right)^n$.

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution.

Suppose first that $\mathcal{I}_1$ has a solution $B_1, \ldots, B_n$. We propose the following solution for $\mathcal{I}_2$:

- we have one interval per task;
- the i-th task is replicated three times and allocated to the set of processors $\{P_u \mid u \in B_i\}$.

We obtain the following reliability for task i:

$$1 - \prod \left(1 - e^{-\lambda\gamma^{a_i}}\right) \geq 1 - \prod \left(\lambda\gamma^{a_i}\right) \geq 1 - \lambda^3\gamma^T \ ,$$

Hence, the overall reliability is $r \geq (1 - \lambda^3\gamma^T)^n$.

Suppose now that $\mathcal{I}_2$ has a solution. We first show that the optimal mapping consists of n intervals, one per task, each replicated 3 times. Suppose that we know the number of intervals in the optimal mapping. There are at most n intervals, and we have enough processors to replicate all of them 3 times, and this increases the reliability. We conclude that all intervals will be replicated 3 times. Suppose now that one of these intervals contains $t > 1$ tasks. There are enough processors to split this interval into t single-task intervals, each replicated 3 times. Let r_1 be the reliability of the original interval with t tasks, and r_t the reliability of the same tasks assigned to t intervals replicated 3 times. By hypothesis of optimality, we have:

$$\begin{array}{lllll}
 & r_1 & \geq & r_t & \\
\Rightarrow & e^{-\lambda\gamma t} & \geq & 1 - (1 - e^{-\lambda\gamma^T})^t & \\
\Rightarrow & \lambda\gamma t - \frac{1}{2}(\lambda\gamma t)^2 & \leq & (\lambda\gamma^T)^t & \text{because } \lambda\gamma^T \leq 1 \\
\Rightarrow & \lambda\gamma 2 - \frac{1}{2}(\lambda\gamma 2)^2 & \leq & (\lambda\gamma 2)^2 & \text{because } \gamma^{T-1} \leq 2 \\
\Rightarrow & \lambda\gamma 2 & \leq & \frac{3}{2}(\lambda\gamma 2)^2 & \\
\Rightarrow & \lambda\gamma 2 & \geq & \frac{2}{3} & \\
\Rightarrow & 4\lambda & \geq & \frac{2}{3} &
\end{array}$$

However, $\lambda \leq 10^{-8}$, which contradicts the hypothesis. This means that, in the optimal solution, any task constitutes an interval.

Let, for all i, $B_i = \{a_j \mid T_i \text{ mapped on } P_j\}$. We obtain the following reliability:

$$r = \prod_{1 \leq i \leq n} (1 - \prod_{a_j \in B_i} (1 - e^{-\lambda\gamma^{a_i}})) \geq (1 - \lambda^3\gamma^T)^n.$$

Suppose that, for a value i, $\sum_{a_j \in B_i} a_j \neq T$. Then,

$$
\begin{aligned}
r &\leq \textstyle\prod_{1\leq i\leq n}(1-\prod_{a_j\in B_i}(\lambda\gamma^{a_i}-\frac{1}{2}(\lambda\gamma^{a_i})^2))\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j}\prod_{a_j\in B_i}(1-\frac{1}{2}\lambda\gamma^{a_i}))\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j}(1-\frac{\lambda}{2}\sum_{a_j\in B_i}\gamma^{a_j}))\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j}(1-\frac{3\lambda}{2}\gamma^{T}))\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j}+\frac{3\lambda^4}{2}\gamma^{T+\sum_{a_j\in B_i}a_j})\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j})(1+\frac{\frac{3\lambda^4}{2}\gamma^{T+\sum_{a_j\in B_i}a_j}}{1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j}})\\
&\leq \textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j})(1+\frac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}})\\
&\leq (1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}})^n\times\textstyle\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j})
\end{aligned}
$$

By hypothesis, we have $\sum_{a_j\in B_i}a_j\neq T$ for a value i. Then by convexity,

$$
\prod_{1\leq i\leq n}(1-\lambda^3\gamma^{\sum_{a_j\in B_i}a_j})\leq
$$

$$
(1-\lambda^3\gamma^T)^{n-2}\times(1-\lambda^3\gamma^{T-1})\times(1-\lambda^3\gamma^{T+1})\ .
$$

By hypothesis, we have:

$$
\begin{aligned}
(1-\lambda^3\gamma^T)^n\leq\ & r\\
\leq\ & \left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n(1-\lambda^3\gamma^T)^{n-2}\\
& (1-\lambda^3\gamma^{T-1})(1-\lambda^3\gamma^{T+1})\\
\Rightarrow(1-\lambda^3\gamma^T)^2\leq\ & \left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n\\
& (1-\lambda^3\gamma^{T-1})(1-\lambda^3\gamma^{T+1})\\
\leq\ & \left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n\\
& \left((1-\lambda^3\gamma^T)^2-\lambda^3\gamma^{T-1}(\gamma-1)^2\right)\\
\Rightarrow\left(\left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n-1\right)\times&\\
(1-\lambda^3\gamma^T)^2\geq\ & \left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n\lambda^3\gamma^{T-1}(\gamma-1)^2\\
\Rightarrow(1-\lambda^3\gamma^T)^2\geq\ & \left(\left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n-1\right)^{-1}\\
& \left(1+\tfrac{\frac{3\lambda^4}{2}\gamma^{4T}}{1-\lambda^3\gamma^{3T}}\right)^n\lambda^3\gamma^{T-1}(\gamma-1)^2\\
\geq\ & \tfrac{1+\frac{3\lambda^4}{4}n\gamma^{4T}}{\frac{3\lambda^4}{4}n\gamma^{4T}}\lambda^3\gamma^{T-1}(\gamma-1)^2\\
\geq\ & \tfrac{1+\frac{3\lambda^4}{4}n\gamma^{4T}}{3\lambda n\gamma^{3T+1}(T-1)^2}
\end{aligned}
$$

However, $3\lambda n\gamma^{3T+1}(T-1)^2\leq 1$ and $1+\frac{3\lambda^4}{4}n\gamma^{4T}\geq 1$. This contradicts the hypothesis. Then, if $\{B_1,\ldots,B_n\}$ corresponds to a solution of $\mathcal{I}_2$, we have $\sum_{a_j\in B_i}a_j=T$ for $1\leq i\leq n$. This shows that $B_1,\ldots,B_n$ is a solution for $\mathcal{I}_1$, which concludes the proof. □

Because mono-criterion reliability optimization is already NP-complete, all multi-criteria problems, with period or latency or both, are also NP-complete on heterogeneous platforms.

7. Heuristics

In this section, we present two heuristics to compute schedules for the multi-criteria problem discussed above. Since we consider several criteria, each heuristic algorithm returns, for a given problem instance, several possible schedules. In the experiments of Section 8, both the period and the latency are bounded; for each instance and each heuristic, we select from the set of computed solutions, the schedule having the best reliability, while still meeting the bounds on period and latency.

Each heuristic consists of two steps: in a first step, the chain of tasks is divided into intervals, and in the second step, the processors are allocated to these intervals. We present the algorithms that are used in these two steps for both heuristics.

7.1. Computation of the intervals

We consider two possible ways to compute the intervals. In both cases, we first decide the number of intervals used, and then we compute intervals according to this value. We can thus compute a set of intervals for any possible number of intervals. Two heuristics are proposed: one aiming at minimizing the period, and a second one aiming at minimizing the latency. The idea is to minimize one criterion, while making sure that the bound on the other criterion is enforced, by selecting the appropriate number of intervals.

In the first heuristic, we try to minimize the latency. Thus, for i intervals, we select the intervals yielding the $i-1$ smallest communication costs. More precisely, for i intervals, we consider the output communication costs of all tasks except the last one. Let $u_1 < \cdots < u_{i-1}$ be the $i-1$ smallest communication costs. Then, the first interval contains tasks τ_1 to τ_{u_1}, the second interval contains tasks τ_{u_1+1} to τ_{u_2}, and so on; the last interval contains tasks $\tau_{u_{i-1}+1}$ to τ_n. This heuristic, denoted Heur-L, is presented in Algorithm 3.

In the second heuristic, we try to minimize the period. We thus try to obtain intervals whose amounts of work are as well-balanced as possible. We use a dynamic programming algorithm to compute the optimal period in the homogeneous case. More precisely, let $F(j,k)$ be the optimal period that can be obtained by grouping the j first tasks into k intervals. The initialization is $F(j,1) = \max\{\sum_{l \le j} w_l, o_j\}$, and the recurrence writes:
$\forall k \ge 2, \forall j \le k$,

$$F(j,k) = \min_{1 \le j' < j} \left\{ \max \left(F(j', k-1), \sum_{j' < l \le j} w_l,\ o_j \right) \right\}.$$

This Heur-P heuristic is presented in Algorithm 4.

Algorithm 3: Heuristic HEUR-L for the computation of the intervals.

Data: a chain of n tasks of size w_i and of output communication cost o_i, a maximum number $\mathcal{K}$ of replications and a number i of intervals
Result: a set of intervals

```
Sort in array A the n − 1 first tasks in increasing order of output
communication cost;
Sort the i − 1 first tasks of A in increasing order of placement in the chain;
The first interval contains tasks from τ_1 to τ_{A[1]};
for j = 2 to i − 1 do
    The jth interval contains tasks from τ_{A[j−1]+1} to τ_{A[j]};
end
The last interval contains tasks from τ_{A[i−1]+1} to τ_n;
```

Algorithm 4: Heuristic HEUR-P for the computation of the intervals.

Data: a chain of n tasks of size w_i and of output communication cost o_i, a maximum number $\mathcal{K}$ of replications and a number i of intervals
Result: a set of intervals

```
for j = 1 to n do
    F(j, 1) = (max{Σ_{l≤j} w_l, o_j}, 1);
end
for k = 1 to i do
    for j = 1 to n do
        Let j' be the value that minimize the function
        x → max{fst(F(x, k − 1)), Σ_{x<l≤j} w_l, o_j};
        F(j, k) = (max{fst(F(j', k − 1)), Σ_{j'<l≤j} w_l, o_j}, j');
    end
end
Let I be an array of size i;
j = n;
k = i;
while j ≥ 1 do
    I[k − 1] = j;
    j ← snd(F(j, k));
end
for j = 1 to i − 1 do
    The jth interval contains tasks τ_{I[j−1]+1} to τ_{I[j]};
end
The last interval contains tasks τ_{I[i−1]+1} to τ_n;
```

Both heuristics produce $\min\{n, p\}$ possible divisions in intervals of the chain of tasks. It remains to allocate processors to these intervals. This is presented in the next section.

7.2. Allocation of processors to intervals

As discussed in Section 5.5, Algorithm Algo-Alloc allocates processors optimally to intervals in the homogeneous case. We use a variant of algorithm Algo-Alloc in the general (heterogeneous) case with a bound P on the period: at the beginning of the algorithm, in increasing order of value $\frac{\lambda_u}{s_u}$, a processor is allocated to the longest possible interval that has no processor allocated to it. Then, step by step for remaining processors, processor P_u is allocated to the interval of greatest value $\left(\frac{\text{reliability with this processor}}{\text{reliability at current step}}\right)$ among the intervals I_j such that $\frac{W_j}{s_u} \leq P$. This is a natural extension of algorithm Algo-Alloc: we first allocate the more reliable processors, and we do not allocate a processor to an interval if the associated computation time exceeds the period bound.

In algorithm Algo-Alloc, it is also possible to take into account allocation constraints preventing some task from being executed on some processor, for instance because some tasks require for their execution a specific hardware driver that exists only on some processors. To do this, we check those constraints before allocating any processor to an interval.

8. Experiments

This section reports experimental results assessing the performance of the heuristics Heur-P and Heur-L, both for homogeneous and heterogeneous platforms. Evaluating only two heuristics may seem too limited. However, to the best of our knowledge, there are no published heuristics that we could compare our results with. In the homogeneous case, Heur-P and Heur-L are compared with the optimal solution computed with the integer linear program presented in Section 5.4. This comparison is quite important, since it allows us to assess the absolute performance of the heuristics, and not just their relative behavior. The heuristics are developed in C/C++ and the integer linear program is implemented with CPLEX [10]. The reader can find the corresponding source code at [14].

8.1. Experiments on homogeneous platforms

To measure the performance of the Heur-P and Heur-L heuristics, we randomly generate applications as follows. For 15 tasks and 10 processors, we randomly generate values of communication and computation costs for 100 different chains of tasks. Then, for a wide range of bounds on period and latency, we compute the number of solutions found within these bounds. We do this for both heuristics and for the linear program. In addition, for the optimal solutions and the heuristics, we compute the average failure probability of the instances where both heuristics have found a solution.

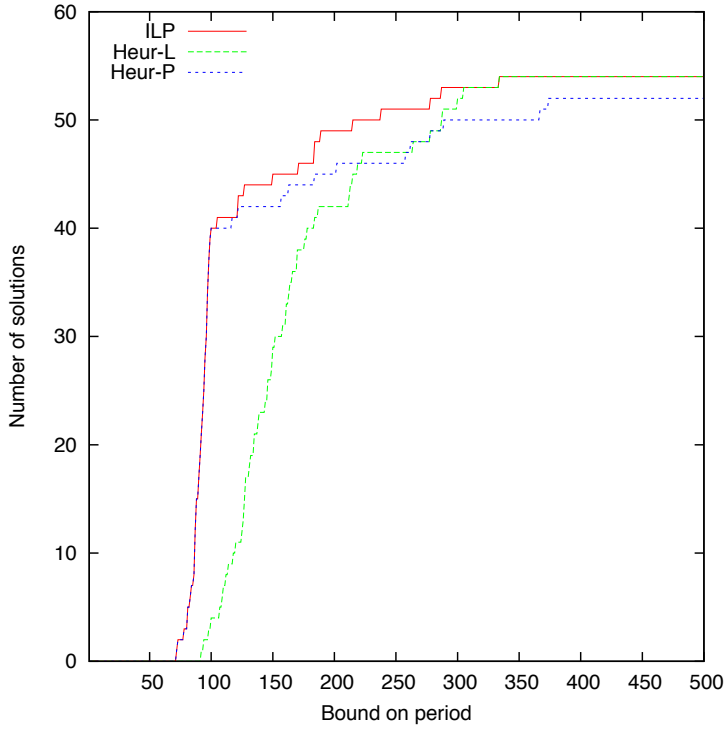

Figure 6: Number of solutions for $L = 750$ on homogeneous platforms.

We set the processor speeds to $s = 1$ computation per time unit, and computation costs of tasks are randomly chosen in the interval $[1, 100]$. The bandwidth is set to $b = 1$, and communication costs are randomly chosen in the interval $[1, 10]$. The failure rate per time unit of processors (resp. communication links) are set to $\lambda_p = 10^{-8}$ (resp. $\lambda_\ell = 10^{-5}$) per time unit. These values are realistic for modern fail-silent hardware components, where one hour corresponds to 100 time units [4] (hence a time-unit is 36 seconds).

With these values, as stated above, 100 problem instances are generated with 10 processors and a chain of 15 tasks. The maximum number of replication is fixed to $\mathcal{K} = 3$. The reasonable period values are found between 70 and 140 time units, and those for latency between 500 and 1000 time units.

In Figure 6, the latency is bounded by 750, and the bound on period is chosen in the interval $[1, 500]$. The value selected for the bound on the latency corresponds to the minimum value such that approximately half of the instances can be solved when there are no constraints on the period. The Heur-P heuristic finds solutions for most of the instances that have a solution, except for high values of the period ($P > 300$). Concerning the Heur-L heuristic, it finds fewer solutions than Heur-P for low and medium values of the period ($P \leq 300$), but

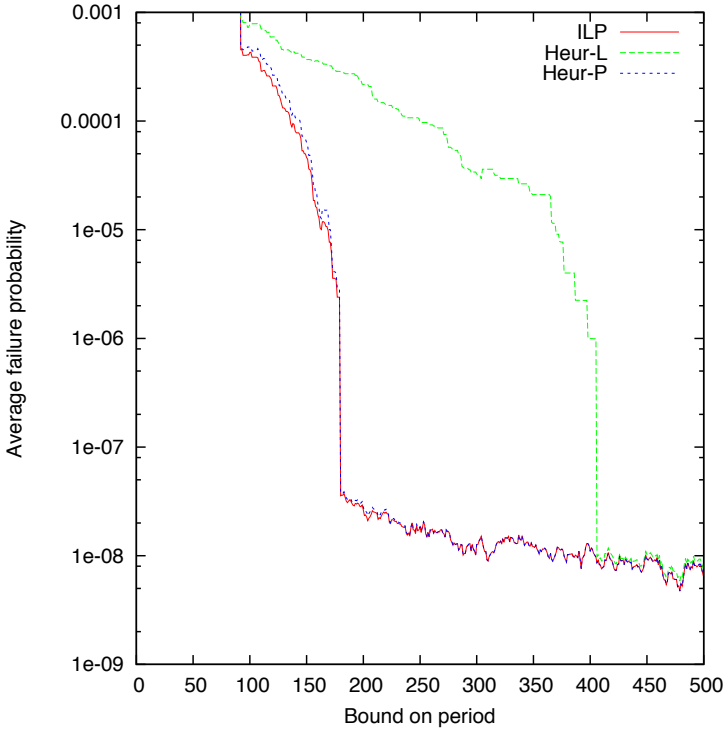

Figure 7: Average failure rate for $L = 750$ on homogeneous platforms.

it obtains more results than HEUR-P, and as many as the ILP program, for high values of the period.

To summarize these results, the HEUR-P heuristic obtains good results in most cases, even though it does not consider latency, hence leading to poorer results when the period is not constrained at all. However, the HEUR-L heuristic becomes efficient for high values of the period, since it focuses on the latency criterion. In contrast, HEUR-L seems to be less efficient than HEUR-P, since it obtains fewer results for reasonable problem instances with a bound on the period. Figure 7 presents the average failure probability of solutions obtained in this experiment. More precisely, it presents the average failure rate of instances where each heuristic has found a solution. We see that most solutions computed by heuristic HEUR-P are very close to the optimal, while solutions of heuristic HEUR-L are less reliable. The abrupt changes of the curve of HEUR-P show that a little increment of the period bound can have a strong impact on the reliability of the solutions.

In Figure 8, we now bound the period by $P = 250$, while the bound on the latency is chosen in the interval $[500, 1100]$. The reason is that no solution is found when $L < 500$, and no additional solution is found for $L > 1100$. In this case, almost all existing solutions are found by both heuristics for low values of the latency. For higher values of the latency, HEUR-P remains efficient, while HEUR-L becomes less efficient. Even without any bound on the latency, HEUR-L fails to find some solutions. This can be explained by the fact that there are more tasks than processors in the instances that we consider (15 tasks and 10 processors). Recall that HEUR-L does not consider the size of intervals, but only the cost of communications. Then for some instances, even with the maximum number of intervals (10 for the instances considered), it can happen that an interval is too large and exceeds the bound on the period.

Figure 9 presents the average failure rate of solutions computed in this second experiment. As in the first experiment, solutions of heuristic HEUR-L are less reliable than solutions of heuristic HEUR-P, and HEUR-P obtains solutions of reliability close to the optimal. We cannot see this difference in Figure 8, which presents similar results for both heuristics.

In Figures 6 and 8, we have compared the performance of both heuristics HEUR-L and HEUR-P for any bound on the period with an average fixed latency, and any bound on the latency with an average fixed period. In the last experiment, we consider the case of a linear relationship between the value of period and the value of latency. In Figure 10, the period is taken in interval $[150, 350]$, and we fix the latency to be $L = 3P$. In this case, almost all solutions are found by both heuristics, regardless of the bound on the period. Note that in most cases, HEUR-P is slightly more efficient than HEUR-L, which confirms our previous observations.

Figure 11 presents the average failure probability for this third experiment. As in previous cases, solutions of heuristic HEUR-P are close to the optimal in terms of failure rate, while HEUR-L obtains less satisfactory results.

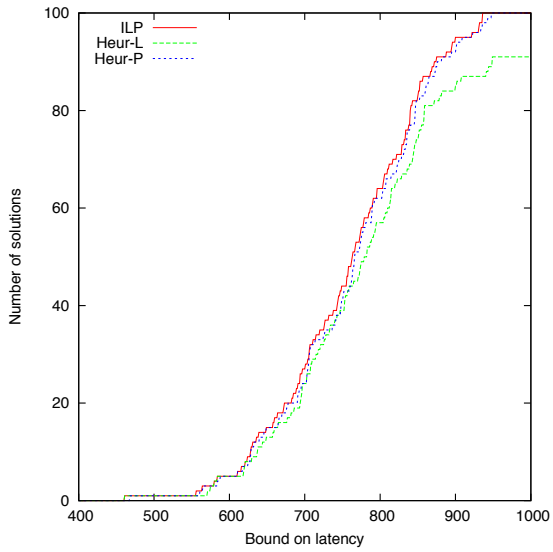

Figure 8: Number of solutions for $P = 250$ on homogeneous platforms.

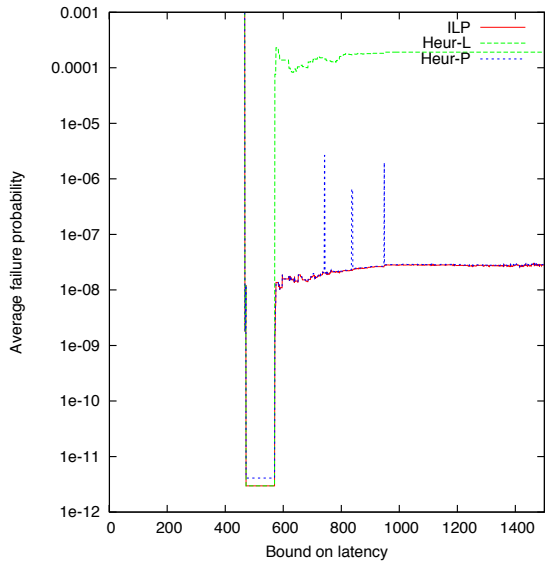

Figure 9: Average failure rate for $P = 250$ on homogeneous platforms.

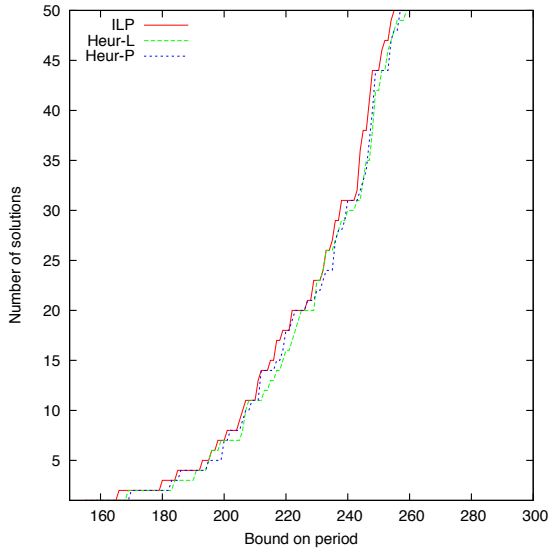

Figure 10: Number of solutions for $L = 3P$ on homogeneous platforms.

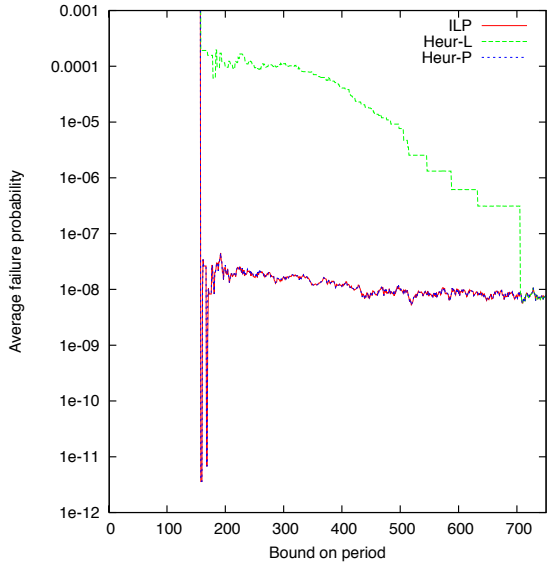

Figure 11: Average failure rate for $L = 3P$ on homogeneous platforms.

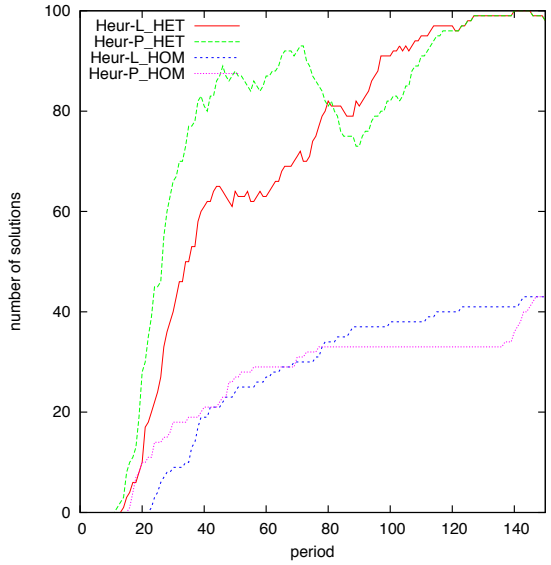

Figure 12: Number of solutions for $L = 150$ on homogeneous and heterogeneous platforms.

8.2. Experiments on heterogeneous platforms

On heterogeneous platforms, we are no longer able to use the ILP to compute the optimal solution. However, we have performed several experiments, using a similar set of parameters as for homogeneous platforms. In each experiment, a chain of 15 tasks, with computation costs randomly chosen in interval $[1, 100]$ and communication costs in interval $[1, 10]$, has to be executed on a heterogeneous platform. Processor speeds are randomly chosen in the interval $[1, 100]$, while their failure rates per time unit have a constant value 10^{-8}. For each plot, 100 platform and application pairs are generated, and we report results for many period and latency bounds. For each experiment, a first instance is created with a chain of tasks and a heterogeneous platform (as described above), and then a second instance is created with the same chain of tasks and a homogeneous platform of speed 5.

In the first experiment, the latency is bounded by 150, and the bound on the period is chosen in interval $[1, 150]$. Figure 12 presents the number of instances for which each heuristic finds at least one solution on homogeneous and heterogeneous platforms. Both heuristics find far more results with heterogeneous platforms than with homogeneous platforms. In this experiment, all instances are solved for large enough period bounds on heterogeneous platforms, while more than half of the instances are never solved (by any heuristic) with homogeneous platforms. Note that on heterogeneous platforms, the number of results is no longer an increasing curve for both heuristics. This is due to the algorithm used to allocate tasks to processors: this algorithm considers only the

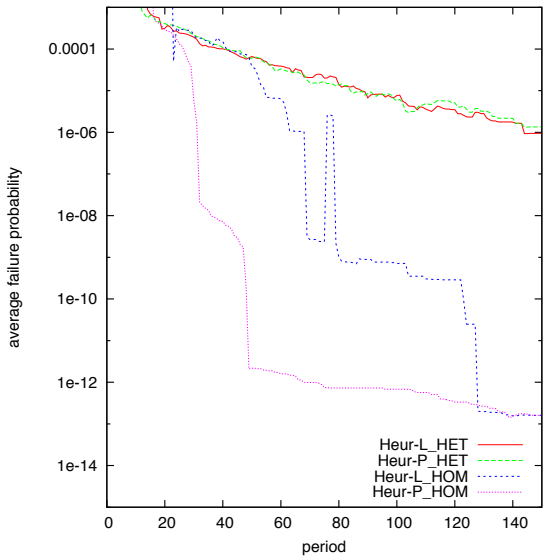

Figure 13: Average failure rate for $L = 150$ on homogeneous and heterogeneous platforms.

period bound, thereby making the sum of interval costs too long for the latency in some cases (while this bound was respected for lower period bounds). As for homogeneous platforms, Heur-P is the most efficient for small bounds on period, and Heur-L obtains the largest number of solutions for large values of the period. The two curves intersect approximatively for the same period bound.

Figure 13 presents the average failure probability of solutions obtained in the first experiment. Note that this implies that for a given value of the period bound, the average values are then not computed on the same set of instances. We see that both heuristics find solutions with similar failure probabilities on heterogeneous platforms. Although both heuristics find more solutions on heterogeneous platforms than on homogeneous platforms, the computed solutions are in average more reliable on homogeneous platforms.

In the second experiment, the period is bounded by 50, and the bound on the latency is chosen in interval $[50, 250]$. Figure 14 presents the number of instances on which each heuristic finds at least one solution on homogeneous and heterogeneous platforms. In this experiment, the plots for heterogeneous platforms seem closer to the plots for homogeneous platforms than in the previous experiment. For a given value of the latency bound, the number of solutions for homogeneous platforms is clearly smaller than for heterogeneous platforms.

Figure 15 presents the average failure probability of solutions obtained in this second experiment. As in Figure 13, the set of instances is not the same for a given value on the latency bound. Here, for latencies greater than 125, heuristic Heur-P obtains more reliable solutions on homogeneous platforms while the other three curves are very close to each other.

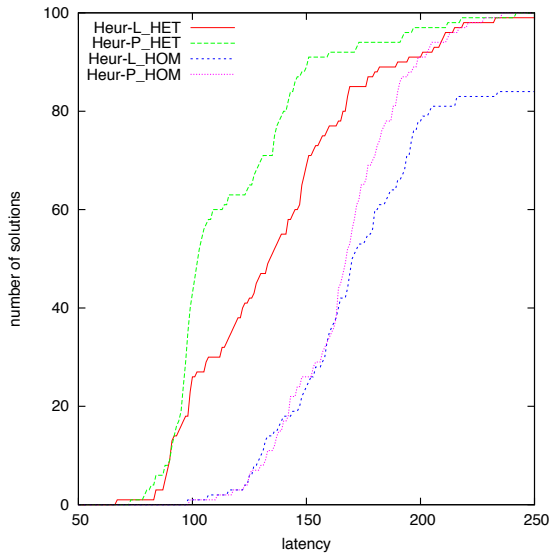

Figure 14: Number of solutions for $P = 50$ on homogeneous and heterogeneous platforms.

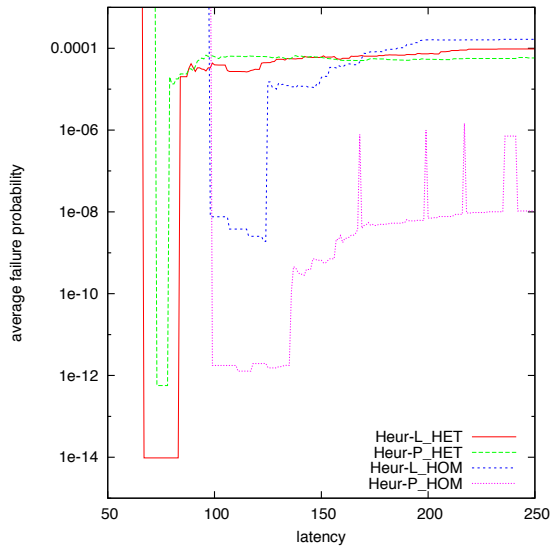

Figure 15: Average failure rate for $P = 50$ on homogeneous and heterogeneous platforms.

9. Conclusion

This paper has addressed difficult multi-criteria optimization problems related to the mapping of pipelined real-time systems onto homogeneous and heterogeneous distributed platforms. The main goal was to optimize the reliability of the mapping through task replication, while enforcing bounds on performance-oriented criteria, namely the period and the latency. The period is a system-oriented criterion, as it measures the aggregate yield of the platform. On the contrary, the latency is a user-oriented criterion, as it characterizes the response time of each individual data set. Period and latency are antagonistic criteria, and achieving good trade-offs for these two objectives is known as a difficult task.

However, performance itself is not enough on failure-prone platforms, and a challenging question is how to achieve application resilience without sacrificing efficiency? But adding reliability to the picture dramatically complicates the mapping problem: one has to decide which computing resources will be kept for performance, and which will be spared (replicated) for coping with failures. Even for homogeneous platforms, deciding how many processors must be assigned to each set of tasks turns out a difficult resource selection problem when three different objectives are considered simultaneously.

Altogether, the results presented in this paper provide a solid theoretical foundation for the study of tri-criteria mappings of pipelined real-time systems. We have derived a comprehensive set of NP-hardness complexity results, together with optimal algorithms for polynomial instances. Another contribution of this paper is the introduction of a realistic communication model that nicely accounts for the inherent physical limitations on the communication capabilities of state-of-the-art processors.

In addition, communication failures have been incorporated in the model through routing operations, which guarantee that evaluating the system reliability remains computationally tractable. An interesting future research direction would be to investigate whether it is feasible to remove this routing procedure, and accurately approximate the reliability of general systems (non serial-parallel).

On homogeneous platforms, we have presented an integer linear program to solve the problem of maximizing the reliability with bounds on period and on latency, while polynomial-time heuristics are derived for the most general problems. We have proposed two heuristics: Heur-L that attempts to minimize the latency and Heur-P that attempts to minimize the period. Our experiments demonstrate the efficiency of the heuristics, and the supremacy of Heur-P in most cases. It would be interesting to address the approximability of the problems with heterogeneous instances, either establishing approximation factors or proving negative results.

Another direction for future work involves the design of heuristics for even more difficult problems that would mix performance-oriented criteria (period, latency) with *several* other objectives, such as reliability, resource costs, and power consumption.

Acknowledgements

We would like to thank the reviewers, whose comments and suggestions greatly helped us to improve the final version of the paper.

This work has been supported in part by the ANR StochaGrid and RESCUE projects. Anne Benoit and Yves Robert are with the Institut Universitaire de France.

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 *Int. Conf. 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. Dependable Secure Comput.*, 1(1):11–33, Jan. 2004.

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

[4] M. Baleani, A. Ferrari, L. Mangeruca, M. Peri, S. Pezzini, and A. Sangiovanni-Vincentelli. Fault-tolerant platforms for automotive safety-critical applications. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03*, San Jose (CA), USA, Nov. 2003. ACM.

[5] A. Benoit, V. Rehn-Sonigo, and Y. Robert. Optimizing latency and reliability of pipeline workflow applications. In *HCW'08, the 17th Heterogeneity in Computing Workshop*. IEEE Computer Society Press, 2008.

[6] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *J. Parallel Distributed Computing*, 68(6):790–808, 2008.

[7] A. Benoit and Y. Robert. Complexity results for throughput and latency optimization of replicated and data-parallel workflows. *Algorithmica*, 57(4):689–724, 2010.

[8] D. Brière, D. Ribot, D. Pilaud, and J.-L. Camus. Methods and specifications tools for Airbus on-board systems. In *Avionics Conference and Exhibition*, London, UK, Dec. 1994. ERA Technology.

[9] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: End-to-end analysis of heterogeneous, multi-resource real-time systems. In *International Conference on Distributed Computing Systems, ICDCS'95*, pages 204–211, Vancouver, Canada, June 1995. IEEE Computer Society.

[10] CPLEX. ILOG CPLEX: High-performance software for mathematical programming and optimization. `http://www.ilog.com/products/cplex/`.

[11] O. Derin, D. Kabakci, and L. Fiorin. Online task remapping strategies for fault-tolerant network-on-chip multiprocessors. In *International Symposium on Networks on Chip, NoCS'11*, pages 129–136, Pittsburgh (PA), USA, May 2011.

[12] 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, Mar. 2002.

[13] J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Symposium on Parallel Algorithms and Architectures, SPAA'07*, pages 280–288, San Diego (CA), USA, June 2007. ACM Press.

[14] F. Dufossé. Source Code for the Experiments. `http://graal.ens-lyon.fr/~fdufosse/reliability/`.

[15] 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 *Int. Workshop on Embedded Software, EMSOFT'01*, volume 2211 of *LNCS*. Springer-Verlag, 2001.

[16] F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, Mar. 1999.

[17] A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Trans. Dependable Secure Comput.*, 6(4):241–254, Dec. 2009.

[18] A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *J. of Parallel and Distributed Computing*, 69(3):326–336, Mar. 2009.

[19] 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, Oct. 2006.

[20] A. Hartman and D. Thomas. Lifetime improvement through runtime wear-based task mapping. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS'12*, pages 13–22, Tampere, Finland, Oct. 2012. ACM.

[21] B. Hong and V. K. Prasanna. Adaptive allocation of independent tasks to maximize throughput. *IEEE Trans. Parallel Distributed Systems*, 18(10):1420–1435, 2007.

[22] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *Euromicro Conference on Real-Time Systems, ECRTS'07*, pages 29–38, Pisa, Italy, July 2007. IEEE Computer Society.

[23] E. Jeannot, E. Saule, and D. Trystram. Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines. In *Euro-Par*, volume 5168 of *Lecture Notes in Computer Science*, pages 877–886. Springer, 2008.

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

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

[26] C. Lee, H. Kim, H.-W. Park, S. Kim, H. Oh, and S. Ha. A task remapping technique for reliable multi-core embedded systems. In *International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS'10*, pages 307–316, Scottsdale (AZ), USA, Oct. 2010. ACM.

[27] C. Liu and J. Anderson. Supporting pipelines in soft real-time multiprocessor systems. In *Euromicro Conference on Real-Time Systems, ECRTS'09*, pages 269–278, Dublin, Ireland, July 2009. IEEE Computer Society.

[28] C. Liu and J. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS'10*, pages 23–32, Sweden, Stockholm, Apr. 2010. IEEE Computer Society.

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

[30] J.-M. Palaric and A. Boué. Advanced safety I&C system for nuclear power plants. In *World Nuclear Congress, ENC'98*, Nice, France, Oct. 1998. European Nuclear Society.

[31] P. Pop, K. Poulsen, and V. Izosimov. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS'07*, Salzburg, Austria, Oct. 2007. ACM.

[32] E. Saule and D. Trystram. Analyzing scheduling with transient failures. *Information Processing Letters*, 109(11):539–542, 2009.

[33] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang-Haeng, and L. Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'12*, pages 71–80, Tampere, Finland, 2012. ACM.

[34] S. 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, Apr. 1989.

[35] J. Souyris, E. Le 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.

[36] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Symposium on Principles and Practice of Parallel Programming, PPoPP'95*, pages 134–143, Santa Barbara (CA), USA, July 1995. ACM Press.

[37] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *Symposium on Parallelism in Algorithms and Architectures, SPAA'96*, pages 62–71, Padua, Italy, June 1996. ACM Press.

[38] TimeSys Corporation. *The Concise Handbook of Real-Time Systems*. TimeSys Corporation, Pittsburgh (PA), USA, 1.3 edition, 2002. http://www.ece.cmu.edu/ ece749/docs/RTSHandbook.pdf.

[39] 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, Nov. 2004.